A Study of Teaching and Learning in Computer Education:

Designing an introductory programming course to foster understanding, problem solving, higher order thinking and metacognition.

by

Margaret J. Kirkwood

A thesis submitted for
the degree of
Doctor of Philosophy
in the University of Glasgow

Volume 1

The Faculty of Arts, Department of Education
University of Glasgow
April 1998

ABSTRACT

This thesis describes the design, implementation and evaluation of a combined learning and research environment for fourteen to sixteen year olds who are learning computer programming at an introductory level as part of an examinable (Scottish Standard Grade) course. The learning environment is designed to foster understanding and the development of pupils' problem solving, higher order thinking and metacognitive skills. It provides an example of how thinking skills can be embedded in academic disciplines (Resnick, 1987) and of how strategies of learning can be acquired in a real context (Nisbet and Shucksmith, 1986).

The teaching method is based upon the following elements: the gradual and systematic introduction of elementary programming concepts, principles and standard techniques; direct teaching on problem solving strategies; modelling of solution processes; on-going formative assessment; and developing metacognition through processes of reflection, articulation and exploration. The interplay between these aspects is analysed to gain a better understanding of how they operate together to promote learning and transfer.

The learning resources were developed within a four-year curriculum development project, the *University of Strathclyde and Lanark Division Programming Project*, which forms an important part of the background to this research. The resources adopt a problem-based methodology and support pupils to work at their own pace.

The research design places an in-depth focus on one class of twenty pupils (the *case study class*) from September 1993 – March 1995 as they are learning to program. The sources of evidence are class records on progress and attainment, folders of work, assessment portfolios, pupil questionnaires and interviews, written reviews, and my informal observations and interactions with participants. Evidence on near transfer is sought through examining pupils' performances on computer problem solving using a spreadsheet, an authentic context. Quantitative and qualitative data analysis methods are combined to identify general trends and individual differences in learners across a range of dimensions.

The findings indicate that the majority of pupils developed a sound knowledge base of programming concepts, principles and techniques which they were able to bring to bear on a range of problems. The problems required the integrated

application of a wide range of higher order thinking skills and the application of a systematic solution strategy. Therefore it can be concluded that the method of teaching programming is sound. The majority of pupils could also reflect insightfully on problem solving processes within and beyond programming, and near transfer was demonstrated in all but a few cases. The learning environment enabled pupils to be very self-directed and to gain better insights into their own learning and thinking processes. Affective responses to learning to program and to the learning environment were generally very positive. However wide differences in progress rates emerged, and a few of the boys responded rather negatively to the course.

There were several constraints such as time, the extent of the programming syllabus, and restrictions imposed by summative assessment requirements. The findings point towards the need for an integrated, cross-curricular strategy to develop learners' problem solving and thinking skills in order to foster transfer. However, overall, I conclude that subject teaching which is focused on developing deep understanding can be successfully combined with teaching directed towards developing generalizable aspects of problem solving, thinking and learning, and that much further work needs to be done in this area.

ACKNOWLEDGEMENTS

I wish to thank both of my supervisors, Hamish Paterson and Dr John Jeacocke, for their interest, strong encouragement and guidance. Eileen, the class teacher of the case study class, provided excellent support, as did her pupils who took the research very seriously and contributed enthusiastically.

The Educational Development Service (EDS) of (the former) Lanark Division of Strathclyde Region and the Faculty of Education at the University of Strathclyde (formerly Jordanhill College) supported the development of the learning resources through funding. Many computing studies teachers in Lanark Division contributed to their development.

TABLE OF CONTENTS

List of Appendices

List of Figures

List of Tables

1 INTRODUCTION

This chapter introduces the research focus and aims, the background to the research, its impetus and methods. It ends with a brief overview of the remaining chapters of the thesis.

1.1 RESEARCH FOCUS AND AIMS

The focus of this research is on the design, implementation and evaluation of a combined learning and research environment for third and fourth year secondary pupils (S3/4) who are learning computer programming at an introductory level as part of an examinable (Scottish Standard Grade) course.

The research design uses a case study of one general/credit class comprising twelve girls and eight boys taught by an experienced and very competent teacher. The progress of the class is followed closely from September 1993 to March 1995 while they are learning to program. During this time the class would study programming for between one (52 minute) period and three periods per week, amounting to seventy hours. Pupils had opted to study computing studies when they made their course choices at the end of second year, and their knowledge of programming was very limited at the start of the course.

The thesis presents detailed evidence, gathered throughout the study, on pupils' progress, attainments, affective responses to learning to program, and reflections on learning and problem solving processes. There is a focus on the individual learner as well as on aggregations, such as average test marks, which are used to indicate general patterns and trends and to examine gender differences. There is also a focus on teaching and assessment processes. The teaching approach is resource-based and differentiated to enable the pace, content and support for learning to be constantly adjusted to suit the needs of the individual pupil. The assessment approach is mainly on-going and formative. The formal requirements for certification are met through a portfolio of evidence.

This study is quite different in character from numerous other studies on computer programming reported in publications such as the *Journal of Research on Computing in Education*, in which an experimental design is used to demonstrate the effects of various 'treatments' (which are rarely expanded upon) on the learners' general cognitive skills, and large amounts of quantitative data are gathered and analysed statistically. I have found that most

of these studies provide little elucidation of the issues surrounding the design of effective learning environments.

The learning and research environment is designed to foster higher order thinking, problem solving and metacognitive skills in addition to an understanding of elementary programming concepts and principles. It provides an example of how thinking skills can be embedded in academic disciplines (Resnick, 1987) and of how learning strategies can be acquired in a real context and taught in such a way as to encourage transfer (Nisbet and Shucksmith, 1986). It further demonstrates the principles of teaching for understanding (Gardner, 1993; Perkins, 1993; Perkins & Blythe, 1994) which Gardner defines as having a sufficient grasp of concepts, principles or skills to be able to bring them to bear on new problems and situations. The research is therefore highly integrative across a range of related themes and important educational agendas, rather than being designed to 'test' a particular theory. It should contribute to an understanding of the inter-relationships between them, in particular, between the specific (subject matter related) and the generalizable aspects.

The teaching method is problem-based, in which a sequence of programming problems provides a vehicle for the systematic introduction of a range of problem solving processes, thinking skills, and programming aspects. The problems have been selected carefully to be varied, natural, interesting, and not too difficult or too easy (Polya, 1948). Any problem contexts which could create an artificial barrier to learning have been avoided, such as problems requiring complex statistical or mathematical processing (such contexts are not untypical in introductory programming courses). As pupils gain in experience and expertise they may tackle more authentic and complex problems.

Direct teaching and modelling of the solution process provide the means to make explicit the problem solving and thinking skills and strategies that are helpful to obtain a solution to a problem efficiently (recognizing, of course, that there may be more than one 'good' solution and more than one 'good' solution path). Through processes of reflection, articulation and exploration pupils are prompted to consider the value of the skills and strategies they have learned and their broader applicability within and beyond programming. These approaches – direct teaching, modelling and developing metacognitive skills (taking the last three together) – should, in combination, enable pupils to transfer skills and strategies more readily from their initial context of learning to new and varied contexts. It should be noted that transfer performances beyond programming are only assessed in relation to one context – learning to

use a spreadsheet – where the focus is on near transfer. These ideas on teaching approaches were explored by Nisbet and Shucksmith (1984; 1986) in a study with primary and secondary teachers which focused on improving children's capacity to think and work strategically. The learning and research environment in this study will provide a further example of how the approaches can be interweaved.

Thus the intention is not just to use problem solving as a better method of teaching programming in order to enhance pupils' understanding of programming, but also to develop in pupils the capacity to solve problems and to think strategically, creatively and critically. This raises a general question: can all of this be achieved in ordinary classrooms? Are these complementary objectives which can operate in synergy or will they compete for attention? Learning and teaching at S3/6 is dominated by the presence of national courses and examinations. These provide less flexibility for teachers because the content and assessment modes are prescribed, and they create additional pressures for both pupils and teachers, particularly in relation to syllabus coverage, meeting assessment requirements and examination performance. Such courses do not provide an ideal test bed for innovative ideas, however in computing studies virtually all of the teaching is delivered through them. It could be argued that problem solving, higher order thinking and metacognition are already present to some degree in the assessable elements of the Standard Grade and Higher Grade computing studies courses, but this may not be reflected in the teaching since the course documentation, issued by the Scottish Examination Board (SEB), contains no advice on teaching methods. Therefore this research has a pedagogical agenda – to enable computing studies teachers to adopt teaching approaches which will enhance pupils' capacities to solve problems, to think and to learn. This agenda need not, and ought not to be restricted to computing studies teachers. The findings may have broader applicability to other curricular areas.

Within the learning and research environment, autonomous learning skills are promoted through a variety of means. The learning resources enable pupils to work independently at their own pace, but it is not intended that pupils should work in isolation from each other or from the teacher. Therefore pupils are encouraged to interact and to initiate collaborations with each other, and some research interventions involve paired or small group activities. Also the teacher has an important role to play as mediator – to guide pupils' problem solving attempts, to develop their thinking and metacognitive skills, and to enable pupils to take responsibility for their learning.

The learning resources were developed within a four-year curriculum development project, the *University of Strathclyde and Lanark Division Programming Project* (which I shall henceforth refer to as the programming project). The work of the programming project forms an important part of the background to this research and is introduced briefly in the next section and discussed more fully in chapter 3. The teacher of the case study class was an effective contributor to the project.

An important aspect of the design of the learning and research environment is how problem solving, thinking and metacognitive skills should be assessed. Which pupil performances will provide the most valid and reliable measures? A range of formative and summative assessment instruments has been constructed for this study (some were developed within the programming project). Certain of these instruments had also to satisfy the SEB summative requirements.

The evaluation of the learning and research environment is wide ranging to encompass the aspects referred to above. The longer timescale of the research provides sufficient time for ideas and innovations to 'bed in', and for the evolution of the case study to be examined, thereby permitting a more accurate and detailed appraisal to be conducted.

1.2 BACKGROUND TO THE RESEARCH

This section fills in the background to the research by discussing those features of computing studies courses in Scotland which are pertinent to the research, the role of problem solving in the secondary curriculum, and the circumstances surrounding the setting up of the programming project (the development of the project and its outputs are described in chapter 3). The discussion on computing studies courses focuses on their evolution, broad content, learning and teaching approaches, and the Standard Grade course.

1.2.1 The evolution of computing studies in Scottish secondary schools

In 1982 the first examinable courses in computing were established in a few secondary schools. By 1993, when HM Inspectors of Schools published a comprehensive report on secondary schools' computing studies courses (HM Inspectors of Schools, 1993) over half of pupils in many schools left with a computing qualification, and most schools offered courses to pupils of all abilities at all stages.

The nature of current provision is generally to offer short courses for all pupils at S1/2, an optional Standard Grade computing studies course or National Certificate modules at S3/4, and an optional Higher Grade computing studies course or National Certificate modules at S5/6. The Certificate of Sixth Year Studies is also offered in some schools. There is a considerable gender imbalance in uptake of optional computing courses; at Standard Grade boys outnumber girls by approximately 2:1 and at Higher Grade by approximately 3:1 (SEB published examination statistics from 1988 to the present).

There is a continuing debate about the computing curriculum and associated teaching methods (e.g. Cole *et al.*, 1990; Cope & Walsh, 1990; Bird, Conlon and Swanson, 1996) which arises from rapid changes in the underlying technology and its applications, because courses are relatively new and still evolving, and because much of the learning about computers can take place within the teaching of other subjects. It is beyond the scope of this thesis to examine most aspects of this debate.

1.2.2 The content of computing studies courses in Scotland

Computing studies involves, 'the study of computers and computer-based technologies, in order to develop knowledge and understanding of computing facts and concepts, associated practical and problem-solving abilities, and appropriate attitudes towards the use of computer technology.' (HM Inspectors of Schools, 1993, p.3) It differs from information technology (IT) mainly because computing studies includes the study of how computers work as well as how they are applied.

The aspects of computing which feature in computing studies courses can be broadly categorized as computer systems, software applications and computer programming. Among the reasons identified by HM Inspectors of Schools for the inclusion of programming are the following:

> Writing and running simple programs allows pupils to control and hence to understand how the computer can manipulate information. By developing programming skills, pupils gain experience of solving problems where they can readily see, evaluate and amend their solutions. Programming skills are required to use some of the relatively new features of general purpose software such as spreadsheet macros. Many pupils, particularly those who wish to undertake academic study of the subject after they leave school, have an interest in developing programming skills. (ibid., p.8)

The problem solving dimension extends beyond programming activity to using other software tools such as spreadsheets, and even to situations where

pupils are not directly involved in implementing solutions, when analysing 'real world' problems, proposing computing solutions (if appropriate) and evaluating computing solutions.

The emphasis on practical problem solving is mirrored in GCSE and A-level computing courses in England and Wales:

> The processes of analysing problems, devising algorithms and carrying through computer solutions to workable and tested conclusions often involve students in very demanding and sustained intellectual activity.
> (Department for Education, 1992, p.21)

1.2.3 *Learning and teaching in computing studies courses in Scotland*

The report by HM Inspectors of Schools examines critically current learning and teaching approaches in computing studies. It emphasizes the role of understanding, and the skills of recalling, classifying, comparing, interpreting, explaining, reasoning, predicting, hypothesizing, reflecting on issues, generalizing and drawing conclusions. It is clear that a 'thinking curriculum' (Resnick and Klopfer, 1989) is envisaged. However the reality observed in some classrooms did not come close to this vision:

> the class listened passively to a teacher describing the design of a program without any direct involvement in discussion about the process. In another school, pupils went straight to the 'what to do' part of the worksheet without reading the explanation ...
> (HM Inspectors of Schools, 1993, p.34)

and:

> pupils were able to produce working code if the problem was simple enough or had been analysed for them, but could not predict what would happen if a simple change was made to the code, or describe how to change the code to solve a similar problem. (ibid., p.34)

Problem solving techniques are also emphasized, and the report discusses three complementary objectives: to enable pupils to solve problems across a range of specific computing contexts; to develop general skills which will enable pupils to solve problems in any computing context; and to develop effective problem solving strategies that can be used across subjects and outside school. In relation to this third objective, the report highlights the need for computing studies teachers to have an awareness of the language of problem solving and of the broad approach to developing problem solving skills used in other subjects. There is however no reference in the report to any initiatives in schools to foster such an awareness.

There is advice, with illustration, on effective teaching methods to develop problem solving skills. Computing studies teachers should develop pupils' knowledge and understanding and problem solving skills in tandem: 'when pupils test a hypothesis about the effect of a particular command, they may be designing and using test data, and evaluating and reporting on the outcome, each of which are problem-solving skills.' (ibid., p.25) Pupils need experience of applying a range of solution stages (such as planning or testing a solution) across different aspects of the subject before they can internalize them and generalize the approaches and use them in different contexts. Pupils, when they are inexperienced at problem solving, also need clear directions on how to tackle each problem and to see good examples of problem solving:

> the process of analysing problems and identifying possible strategies for solving them is difficult, and it is helpful to take pupils through these processes before asking them to analyse a complex problem without teacher assistance. (ibid., p.26)

Classes need to be organized in a number of ways to promote effective learning from problem solving exercises by, for example, encouraging class or group discussion. Once again, however, there is evidence of less successful approaches in use:

> there is a tendency for pupils to work through a series of detailed exercises without ever developing a conscious strategy for solving programming problems. (ibid., p.34)

and:

> There is still a tendency for some teachers to focus on specific programming techniques and neglect more general issues, and for others to adopt an approach that concentrates almost entirely on general methods without giving pupils enough structured experience of specific techniques. (ibid., p.34)

The over-emphasis on specific techniques is also criticized by HM Inspectors in England and Wales:

> some courses simply require them [students] to learn in detail how to use particular pieces of software, often without understanding what kinds of problems such software was designed to solve. If IT is to be treated in a practical way then clearly it is necessary for pupils to learn how to use particular packages, but it should be recognized that software has only temporary currency ...The emphasis should, therefore, be on the broad nature and functions of IT tools, the tasks for which they are useful and consequent selection of what is suitable.
>
> (Department for Education, 1992, p.23)

Independent learning is seen by HM Inspectors in Scotland as the main contribution of computing studies to general learning skills:

> Independent learning involves planning, making choices, taking responsibility, seeking comments and undertaking self-evaluation. These skills need to be taught and encouraged, and good computing teachers teach them explicitly. (HM Inspectors of Schools, 1993, p.31)

Some teachers seem to take for granted that their pupils possess certain learning techniques, such as ways of keeping notes that will aid later revision or organizing their time:

> while they may criticize poor practice or help specific individuals who had difficulties, they did not actively teach these learning techniques to their classes. (ibid., p.31)

The need for teachers to respond to the needs and abilities of individuals is stressed. Learning and teaching approaches and course content should be flexible enough to cope with the range of previous experience and attainment of pupils. Approaches where all pupils are required to undertake exactly the same tasks are strongly criticized:

> This can result in the more able pupils spending much of each lesson doing exercises as part of a whole-class lesson that are trivial for them and which, in some cases, they have previously covered on their own, while weaker pupils struggle through with tasks that may be incomprehensible to them. (ibid., p.36)

Even when pupils are enabled to work at their own pace through a common set of activities, this can still mean that all pupils must work through the most elementary materials, whether or not they need to.

Low motivation, and little learning of the underlying computing principles was observed to occur in the following circumstances:

> where pupils were inadequately prepared for what they were about to do, where they followed instructions from worksheets mechanically at work stations, and where there was limited variety from day to day ... (ibid., p.37)

Some of the most productive learning was observed to arise from well-considered questions such as, 'What would happen if ...?' or 'How could you get the computer to ...?', which one can link again to the concept of the thinking curriculum.

1.2.4 *An overview of the Standard Grade Computing Studies course*

The preceding discussion on learning and teaching is relevant to the Standard Grade computing studies course which is now in its third version – the Amended Arrangements (SEB, 1991) – having been reduced in scope, updated and simplified from the original course which was fully introduced (following limited piloting) in 1986.

1.2.4.1 *Aims and topics*

The aims of the current course are as follows: to develop knowledge and understanding of computers, how they operate, and their effect on the individual and society; to develop the ability to apply knowledge and understanding of computing facts in order to find solutions to problems; and to develop practical abilities in the use of computers and computer software in order to solve practical problems. The syllabus topics are: computer systems (20 hours), computer programming (40 hours), project work (30 hours), and computer applications (60 hours).

1.2.4.2 *Key features of the assessment model*

The assessment model at Standard grade is rather complex and difficult to explain without jargon. A feature of all Standard Grade courses is a certificate which records for each candidate an overall award on a 7-point scale of grades (1 being the highest), and a profile of awards for distinct assessable elements of the course. The 7-point scale maps onto three levels of award (and 'course completed') as follows: grades 1, 2 – credit; grades 3, 4 – general; grades 5, 6 – foundation; grade 7 – course completed. Within a level, the upper grade indicates a 'high overall standard' of performance and the lower grade a 'satisfactory overall standard'. For each assessable element there is a detailed set of criteria (the extended grade related criteria) to enable assessment instruments to be designed at levels and to guide the process of grading pupils' work.

In computing studies the three assessable elements are:

knowledge and understanding (KU) – recall, describe and explain computing facts and concepts, as indicated in the content sections of the Amended Arrangements. The content sections are set out in a columnar arrangement as core (foundation), extension (general) and further extension (credit);

problem solving (PS) – analyse problems and design and evaluate solutions in a variety of theoretical contexts;

practical abilities (PA) – analyse problems and design, implement and evaluate solutions in a variety of practical situations.

The main component skills in PS are identified in the extended grade related criteria as: describing the fundamental nature of the problem and identifying its main features; describing the constituent processes to obtain a solution and the sequence in which they should be carried out, recognizing where standard techniques or existing software could be used; describing how the main solution stages can be developed; explaining why a particular approach is adopted; assessing the adequacy of a solution; considering its implications; and suggesting improvements to a proposed solution or solution strategy.

The main component skills in PA are identified as above, with the addition of a range of implementation skills: obtaining information from a variety of sources; using hardware and software tools effectively and efficiently; identifying and rectifying errors; writing documentation; and reporting on and discussing progress.

KU and PS are externally assessed via a written examination at the end of the course (there is one examination at each level covering both elements). PA is internally assessed via a project and four items of coursework. The project can be based on any syllabus topic and is selected by the school for each pupil from a bank of projects that is issued by the SEB. Some projects are designed as core and extension and straddle two levels, e.g. general/credit, and others are designed as single level. A detailed marking scheme is issued with each project. The coursework tasks and marking schemes are generated by the school (marking, rather than direct grading, is the recommended procedure). A programming coursework grade (based upon two items of fully-documented evidence) and a non-programming coursework grade (also based upon two items) are awarded.

The two main factors which enable the teacher to discriminate between performances at different levels are the complexity of the task and the degree of support given to the pupil to accomplish it. To support teachers in making judgements about these two factors, a set of exemplar coursework tasks and graded evidence was issued to schools by the SEB (1993b), along with general guidance on procedures.

The project, programming coursework and non-programming coursework grades are then averaged to obtain the PA grade. The coursework tasks, marking schemes, and a selection of all of the evidence (based upon a stratified sample across the range of grades awarded) are centrally moderated by the SEB,

using a sample of schools each year. The KU, PS and PA grades are combined in the ratio 1:2:2 to obtain the overall award. Thus problem solving abilities, whether demonstrated in theoretical contexts or in practical situations, account for 80% of the overall award.

The titles of two of the assessable elements – problem solving and practical abilities – are somewhat misleading. It would be reasonable to assume that problem solving skills are assessed under the PS element and therefore make up 40% of the overall award, and that PA assesses something other than problem solving. It would also be reasonable to assume that a problem solving element encompasses all important aspects of problem solving, but it does not since implementation skills are not covered, enabling PS to be assessed via a paper and pencil test. Indeed the author of a review on problem solving in the secondary curriculum fails to notice the presence of a second 'problem solving' element – the PA element – in computing studies (Holroyd, 1989). However the review highlights precisely the same anomalies in the four sciences (science, biology, chemistry and physics) which use the same element structure, and it is clear that some central steering has resulted in this pattern across subjects. If, as HM Inspectors of Schools (1993) suggest, computing studies teachers should have an awareness of the language of problem solving and of the broad approach to developing problem solving skills in other subjects then the element structure within subjects should not obscure the presence of problem solving in different aspects of the course.

1.2.4.3 The programming topic at Standard Grade

The introduction to the programming topic states: 'Programming is concerned with the development of the practical abilities of problem analysis and design of solutions, practical implementation (which includes coding) and evaluation.' (SEB, 1991, p.7) Programming is thus being construed as a practical, problem solving activity.

The programming content is set out in a *general programming specification* which indicates the main aspects – representation of solution, sequence, modularity, simple data types, repetition, and so on – and items of content at each level. The school can use any programming environment(s) which will provide 'broad coverage' of this content (a range of types of programming environment is suggested), and it can decide on which language features to introduce at each level so long as broad compatibility is maintained with the specification.

Programming is not assessed in the external examination but through programming coursework, and also via the project if the pupil opts, or is allocated to do an assessed project on programming rather than on a different syllabus topic. The written evidence for each of the two items of programming coursework comprises: an analysis; the structure of the program; a printed listing; some sample runs (where appropriate) as evidence of testing; and an evaluation of the solution.

1.2.5 *The University of Strathclyde and Lanark Division Programming Project*

This project was set up in response to the concerns of computing studies teachers in one local authority (the former Lanark Division of Strathclyde Region) about teaching and assessing the programming topic at Standard Grade. It was funded by the Educational Development Service of Lanark Division for four years (1990-4) and I was invited by the teachers, through the adviser in educational computing, to direct the project. Over the duration of the project I was contracted to work directly with teachers for 64 days, with a further 64 days being allocated for preparation, development of resources, administration, and so on. The educational adviser, two of the teachers and I formed a planning group to identify potential teacher participants. The group of eleven teachers who joined the project in May 1990 were from the computing departments of six urban secondary schools. The intention was that the number of participating teachers and departments would increase gradually over the duration of the project.

The following observations were made by the teachers at the first project meeting, which focused on identifying the problems:

- teaching was not well differentiated, i.e. there was insufficient adjustment to the pace, content or support of programming work to cater well for individual learning needs;
- the organization and management of programming classes was difficult (and this was exacerbated by equipment problems);
- the assessment of practical coursework was complex;
- there was too much content to be covered, particularly at credit level;
- many pupils' understanding of programming concepts and principles was poor;

- many pupils' problem solving skills were very poorly developed, and they could not apply their knowledge of language features or programming techniques to solve novel problems;
- the range of higher order thinking skills that programming demands seemed beyond many pupils' capabilities;
- many pupils were very inept at managing their learning; and
- (not surprisingly) many pupils showed very little enthusiasm for programming.

(from notes taken at the first project meeting on 21/5/90.)

Many of the problems were attributed by teachers to the unsuitability of the learning resources since, in order to cater for a range of abilities in the class, they were using a hotch-potch of different materials – some designed for foundation level only, some designed for use as part of a whole class lesson but a group of pupils was having to work through them on their own, and all in different formats (worksheets, booklets, textbooks) and employing different teaching methods. Also none of the resources placed sufficient emphasis on problem solving skills or supported pupils to manage their learning effectively. Whole class teaching had not been found to work either, mainly because teachers found it impossible to pitch the lesson accurately.

There were, however, many favourable circumstances surrounding the project, in particular:

- The project was self-initiated, which demonstrates the participants' willingness to be openly self-critical and their commitment to improving their teaching.
- The Standard Grade syllabus was under review – the educational adviser was chairing the review group – and it was hoped that some of the content and assessment demands would be reduced.
- The teachers had a wealth of experience between them, the majority having computing as an additional teaching qualification, which enabled the group to draw on knowledge of teaching methods across a range of subjects including science, mathematics and Latin.
- The teachers felt that the programming environment in current use in all Lanark Division schools (Acornsoft COMAL) provided a satisfactory

vehicle for secondary pupils to learn basic programming concepts and principles.

At an early stage of our deliberations we agreed on the characteristics of the learning environment that we would seek to create: all pupils should have a positive experience of learning to program; learning should be self-paced and supported by resources that would be developed within the project; problem solving skills should be emphasized in the teaching; pupils should be given support to manage their learning effectively; flexible approaches should be used to cater for individual learning needs within a coherent programme of study; and effective manageable systems should be put in place (e.g. to monitor pupil progress and to gather assessment evidence).

The project was also concerned with the professional development of the teacher participants, and beyond this to all computing studies teachers in Lanark Division through an on-going programme of dissemination. The broad range of learning, teaching and assessment issues to be addressed by the project would have applicability beyond programming to all aspects of the computing studies curriculum. The subsequent development of the project and its main outputs (specifically the learning resources) are discussed in chapter 3.

1.2.6 Problem solving 'across the secondary curriculum'

In a review of problem solving in the secondary curriculum commissioned by the Scottish Education Department (Holroyd, 1989) the author begins by observing that the terms 'problem' and 'problem solving' are rarely defined by the people who use them, and yet they can mean many different things to different people. Having examined a range of possible definitions the review concludes that there is no one satisfactory, all-encompassing definition, and it is therefore important for people to clarify the meaning that they attach to the terms. (This is a matter that I shall attend to in chapter 2.)

The review then examines the place given to problem solving in the secondary curriculum according to various official reports and the separate Standard Grade Revised Arrangements documents covering twenty-seven subjects. Permeation to the whole curriculum at all secondary stages is the message which emerges from official reports. For example, in *Curriculum Design for the Secondary Stages* (Scottish CCC, 1987) problem solving is described as a process skill essential to every pupil's development which should permeate all modes, courses and activities, is explicitly mentioned in three of the eight curricular modes at S1/4 (mathematical, scientific and technological), and is viewed as a component of creative thinking at S5/6.

In relation to Standard Grade courses, problem solving features strongly in nine course descriptions (including computing studies). The other courses could easily be re-written with problem solving in a conspicuous position. An example is given by Holroyd (1989) of an investigation in history: 'What was it like to be a servant in Victorian Glasgow?', which could be re-cast in an alternative wording, without distortion: 'Your problem is to think yourself into the situation of a servant in Victorian Glasgow. What was it like to be that person?' etc. (p.24)

When problem solving is given an important place in syllabus documents, this can be for a variety of reasons:

(i) Problem solving is an important feature of the paradigm within a discipline: to be inducted into a way of knowing, one has to experience the characteristic modes of enquiry. This is an epistemological argument.

(ii) Problem solving is the means by which understanding of the subject is enhanced (it is a means to an end).

(iii) Problem solving is important because experience of solving a range of problems develops generalizable problem solving skills (it is a useful end in itself).

(iv) Problem solving can make a subject seem real and challenging, and successful problem solving is reinforcing. This is an argument about motivation.

Of course, it cannot be assumed that all syllabus intentions will be reflected in classroom practices, particularly since SEB syllabuses set out the aims, content and assessment arrangements but do not provide advice on teaching approaches. Drever (1988, p.86) argues that such advice is needed because of the shift in emphasis, with the introduction of Standard Grade, from content to skills and from products to processes. He comments:

> Various curriculum documents stress that teachers will need in-service training to develop new skills, but the skills are ill-defined add-on extras, with no discussion of the skills teachers are supposed to have already and how these might be developed.

The review by Holroyd is useful to highlight some of the potential problems of using permeation to develop generalizable problem solving skills, specifically: there is not a common language to describe problem solving and therefore

some problem solving activities are hidden under different titles (such as investigation or experimentation); problem solving serves a range of purposes across different subjects and some purposes may get squeezed out, for example, when covering content is the main prerogative; pupils may get confusing messages about strategies from different subject areas; and the evaluation of the effectiveness of permeating aspects is notoriously difficult. To this can be added Drever's observation on the need for a more precise formulation of the skills that teachers will need to develop problem solving skills in pupils.

It is evident from Holroyd's report that too little thought has been given in the past to how problem solving is embodied in the overall design and delivery of the secondary curriculum at S3/4. In the context of this research, where pupils are being asked to reflect on learning and problem solving processes, it will clearly involve considerable mental work for them to do this since there is no clear articulation of problem solving 'across the curriculum'.

Current proposals for the post-16 curriculum in Scotland list communication, numeracy, problem solving, information technology and working with others as essential core skills viewed as transferable across subjects and contexts (Higher Still Development Programme, 1997). Within this framework, problem solving has the following components: critical thinking, planning and organizing, reviewing and evaluating. Any opportunities for the assessment and automatic certification of core skills within existing courses will be identified in advance, resulting in no additional work for the student or teacher. Will this programme have any effect on how students experience problem solving across the curriculum or on their level of skills development? If all that it does is to audit the content of existing courses it seems unlikely that there will be any changes for the better.

1.3 IMPETUS

This section addresses two questions: what has led to my interest in the research area, and why is the topic important and worthy of investigation? I present a brief, chronological account of how my interest developed in the teaching of problem solving within programming and also more generally in curriculum development. I then discuss the general importance attached by others to fostering problem solving, higher order thinking and metacognitive skills in learners, and some views on how the education system can be changed to bring this about.

1.3.1 *What has led to my interest in the research area?*

I first encountered a problem solving schema – based on Polya's (1948) four phases of problem solving – while seconded as the development officer for the *Feasibility Study for Foundation Level Mathematics* in 1980-1. The attempt to place problem solving at the heart of the mathematics curriculum represented, at that time, a radical shift in pedagogy and assessment. Having returned to mathematics teaching, and with the advent of microcomputers into schools in the mid-1980's, I introduced all pupils in my classes to programming in Logo and BASIC as a vehicle for teaching problem solving, for learning mathematical concepts, for fun and exploration, and to familiarize pupils with using computers.

As a new lecturer in the computer education department at Jordanhill College of Education, I chose to develop a programming course in COMAL at foundation level (Kirkwood, 1986) as my contribution to a curriculum development project to support the introduction of the Standard Grade course in computing studies. The programming course was problem-based and individualized to enable pupils to work at their own pace. It was widely adopted by schools at that time. A secondment from 1986-8 as the national development officer for Standard Grade computing studies brought close contacts with many teachers, teacher educators, and educational advisers. Programming emerged as one of the most contentious areas of the course[1], and concerns were expressed about the lack of suitable learning resources for teaching programming to the full ability range.

I was introduced to Stenhouse's (1980; 1983) ideas on curriculum development while studying for an M.Ed. degree at the University of Glasgow. His views on how teachers must engage with curricula resonated strongly with my experiences of working with teachers through inservice, where I often wondered, did any of the ideas discussed on the course ever find their way into classrooms? He uses a slogan, 'No curriculum development without teacher development':

> But that does not mean ... that we must train teachers in order to produce a world fit for curricula to live in. It means that by virtue of their meaningfulness curricula are not simply instructional means to improve teaching but are expressions of ideas to improve teachers. Of course, they have a day-to-day instructional utility: cathedrals must keep the rain out.

---

1 The debate on the relative merits of different programming languages – Logo, COMAL, BASIC, Pascal, Prolog – was conducted through the pages of the Scottish Times Educational Supplement.

> But the students benefit from curricula not so much because they change day-to-day instruction as because they improve teachers.
> (Stenhouse, 1983, p.156)

This implies that curriculum development in itself constitutes a process of teacher development (Elliott, 1991). Stenhouse also speaks of the theory-practice divide and of curricula as the bridge between the two.

> All educational ideas must find expression in curricula before we can tell whether they are day dreams or contributions to practice. Many educational ideas are not found wanting, because they cannot be found at all. (ibid., p.158)

The opportunity to direct the programming project provided an ideal context to bring all of these interests together. Although the main focus of this research is not on the conduct of the programming project itself, involvement in the project assisted me to identify the key issues and themes that the research examines and explores by means of the case study. My initial focus on problem solving broadened out to include the development of higher order thinking skills and learning to learn as I began to engage with the project, to read more extensively and to discuss with my supervisors.

The timescale of the programming project (1990-4) and of the case study (1993-5) overlap. For the thesis to have placed an in-depth focus on the programming project in all its dimensions – teacher professional development and improving the learning environment for pupils across a very wide range of aspects – would have resulted in an unmanageable writing task. Therefore aspects of the programming project are covered (in this chapter and in chapter 3) as an important background to the case study.

1.3.2 Why is the topic important and worthy of investigation?

The report by HM Inspectors of Schools (1993) provides support for an embedded approach to developing problem solving, thinking and learning skills in computing studies, albeit that its main focus is naturally on the development of subject specific understanding and competence (it is one of a series of reports by HM Inspectors of Schools each on different subject areas of the secondary curriculum). The programming project teachers' observations on the many problems that they faced when attempting to teach and assess programming at Standard Grade are echoed in the section of the report which provides a critique of learning and teaching. This signifies that the problems were not confined to this particular group of teachers in the circumstances that they found themselves when the programming project was set up.

From the review of problem solving in the secondary curriculum (Holroyd, 1989) it is clear that the approach to embedding problem solving into academic subjects at Standard Grade is not well developed, and therefore one would predict that a similar set of problems would be experienced in other subjects which might stimulate developments in these areas, creating the potential for cross-fertilization of ideas. One good example is in practical experimentation in chemistry, where Hadden (1991) has provided a critique of conventional, 'recipe following' teaching methods and has developed and trialled a set of projects designed to engage Standard Grade pupils in 'genuine' problem solving 'at the bench'.

The need to create and research into learning environments to foster understanding or general cognitive skills is a strong theme in the educational literature (which I shall discuss in chapter 2). For example, Nisbet and Shucksmith (1984, p.16) comment on the sorts of classroom conditions under which metacognition is unlikely to take place, and therefore, by implication, on the sorts of conditions under which it is:

> Consider the situation where little or no conscious thought is required to process instructions and perform a task; where you are not stretched by new material or problems; where you are not required to assess and monitor your own progress or performance; where the principal demands are for 'busy' work in an overcrowded examination syllabus. Unfortunately, this sounds depressingly like a description of the average school classroom.

Conditions favourable to metacognition could be identified from this as: work which challenges pupils and demands some thinking and effort; the requirement for pupils to regularly assess and monitor their own progress and performance; and the provision of worthwhile tasks which advance the learners' knowledge, understanding and competence, with sufficient time being made available to accomplish them (Kirkwood, 1995).

This is not to suggest that it would be easy for individual teachers to integrate ideas on learning strategies and metacognition into their practice. Nisbet and Shucksmith (1986, p.94) identify that perhaps the most daunting aspect for teachers is that the ideas do not come pre-packaged:

> Any teachers currently interested in the ideas must sift through the material, preparing guidelines for themselves that best match their own teaching styles, that suit their teaching aims and that will best benefit the type of children with whom they are involved. Perhaps only when the ideas become incorporated into curriculum objectives and structures and 'whole-school' policies are developed will it be possible to assess the impact of the ideas fully.

English language publications, keyword in title	1997 (Jan – Sep)	1992 – Sep 97
problem solving	21	784
thinking	30	1245
metacognition	1	42
understanding	38	1481
	90	3552

Table 1: Number of retrievals on keyword searches on the title of English language publications, using ERIC on the Internet, 4/12/97.

Australian, British, & Canadian publications: keyword in title	1976 – Sep 97 (Australian from 1978)
problem solving	1151
thinking	1379
metacognition	89
understanding	1797
	4416

Table 2: Number of retrievals on keyword searches on the title of publications, using International ERIC on CD-ROM, 4/12/97.

By which processes will the ideas become incorporated into curriculum objectives and structures and 'whole-school' policies? Perhaps it is partly *through* the inquiry of interested teachers that this will happen. Fullan (1993) speaks of fruitful conditions for change in educational systems where simultaneous bottom-up and top-down initiatives merge, and Elliott (1991) speaks of educational theories being validated *through* practice rather than being validated independently and then subsequently applied to practice.

1.4 METHODS

This section discusses my engagement with the research literature and introduces some aspects of the research design.

1.4.1 Engagement with the research literature

I sought a careful, systematic and thoughtful engagement with the literature to enable me to build a coherent account of theories and research. Three broad areas of reading were identified. The first area was on the 'thinking curriculum', teaching for understanding, teaching for transfer, 'learning to learn', and the affective and social dimensions of learning and thinking. The second area was on computer programming in schools. The third area was on research design and research methods relevant to this investigation.

1.4.1.1 The first area – the 'thinking curriculum' etc.

To have attempted from the outset to retrieve texts by electronic means (using ERIC and International ERIC[2]) on the keywords 'problem solving', 'thinking', 'metacognition', and so on, would have resulted in an unmanageable number of titles with no ready means of sifting them. This did not seem to be the best initial course of action. To illustrate this, I have conducted a straightforward search on ERIC, accessed on the Internet today (4/12/97), on the following keywords: 'problem solving'; 'thinking'; 'metacognition'; and 'understanding'; to appear in the title of English language publications (table 1). There is a much larger number of retrievals for the keyword appearing in any part of the entry e.g. 6543 for 'problem solving' and 9526 for 'understanding' over the period 1992 – Sep 97. The outcome of a further search on International ERIC, accessed on CD-ROM, is indicated in table 2.

---

2 ERIC – US Government Educational Resources Information Center. International ERIC – Australian, British and Canadian Educational Indexes.

Instead I began with Nisbet's SCRE Spotlight (1990) entitled *Teaching Thinking: an Introduction to the Research Literature*, and Holroyd's review of problem solving in the secondary curriculum (1989) which contained a list of references, a chapter on theoretical background encompassing significant sources and a chapter on recent work in Scotland.

The SCRE Spotlight led me to texts and/or articles by Resnick (on thinking skills), Nisbet and Shucksmith (on metacognition and learning strategies), De Corte (on powerful learning environments), Boekaerts (on emotion, motivation and learning), Flavell (on metacognition), Papert (on problem solving and thinking skills development within Logo programming environments), Perkins and Salomon (on cognitive skills and transfer), Brandt (on a range of topics relevant to this research, in the form of conversations with those at the leading edge of their fields), and others. Some of this reading related directly to the teaching of programming in schools e.g. Papert, and an article by Salomon and Perkins (1987) on transfer of cognitive skills from programming. All of these provided well written, authoritative, often very interesting and accessible accounts of theories and research relevant to classroom practice which contained many other useful references, e.g. to Polya and Schoenfeld (on mathematical problem solving) and Gardner (on teaching for understanding).

Some of the references from Holroyd's review were more difficult to engage with. Many were very subject focused and not really intended for a general audience e.g. on science teaching (Millar and Driver, 1987), others were some distance from the classroom, on theories of human problem solving behaviour derived from analyses of machine 'problem solving' behaviour, and others so comprehensive that I felt overwhelmed e.g. Frederiksen (1984).

I made some use of electronic searches on ERIC and International ERIC, for example, using author searches and when trying to locate work on a very specific theme or topic, such as the output from a particular project. As someone with a computing background I have no difficulty with using such resources to conduct searches. I also used the Jordanhill library computer system, the list of journals held by the library, the British Educational Index (BEI) and inter-library loans to locate and gain access to materials.

Contacts with people – my supervisors, at work, at conferences, seminars and workshops – were the other vital sources of guidance on appropriate reading in this area.

1.4.1.2 The second area – computer programming in schools

This area splits into two rather distinct bodies of literature, one centred on Logo programming which is very child centred in its philosophy, and the other on programming courses (on a variety of languages – BASIC, Pascal etc.) within secondary or tertiary education which is very computing science oriented.

In the Logo tradition the programming language is an expressive medium and not the intended object of interest, giving children the right to be the makers, and not merely the users, of computer knowledge[3]. A very good example of research which reflects this philosophy is provided by Kafai (1993). She gives a fascinating account of how sixteen fourth-graders (at primary school) each spent six months on designing and producing a computer game to teach third graders about fractions.

In relation to programming within schools' computing courses, I have focused on methods of teaching rather than on specific features of languages or development tools (although I appreciate that the two areas are not divorced from each other – it is just a question of emphasis). An article by Linn and Clancy (1992), on teaching programming through case studies which incorporate solution commentaries by expert programmers, provides a rigorous justification for the case study method which is grounded in the recent research on teaching for understanding and developing problem solving skills in computing and other subjects (but here the focus is less on generalizable skills). It is therefore a very useful article to map out some of the territory.

There are numerous studies assessing the impact of programming on cognitive skills. Most are based on experimental designs, and a common failing is that there is insufficient explication of the teaching methods, the conditions being compared or the assessment instruments. Salomon and Perkins (1987) and Keller (1990) describe the results of such studies as mixed. Given the range of cognitive skills being investigated, the age range of students (from pre-school to under-graduate), and the range of teaching methods or conditions, this hardly seems surprising. However it is possible to discern some patterns in the findings on transfer from programming, with positive effects being associated with strong teacher mediation (Salomon and Perkins, 1987; Keller 1990) and the effective use of peer interaction (Keller, 1990).

---

3 Taken form my notes of an address by Seymour Papert at the International Conference on Technology in Education at MIT in March 1993.

1.4.1.3 *The third area – research methods*

Comparisons of research designs, collaborative action research, case study design, research ethics, and methods of data collection, assessment and evaluation were the five main focuses of reading. Texts and articles on research methods and issues were one obvious source by, for example, Stenhouse (1983), Elliott (1991), Nisbet (1988), Drever (1995), Harlen (1994), Munn and Drever (1990), Baron (1987) and Coolican (1994). Fullan (1993) provides a very philosophical treatment of change processes in education and schools which is relevant to this work.

Another very useful source were those writers who explained and justified carefully their chosen designs, such as Papert (1980; 1994), Harlen and Malcolm (1997), Schoenfeld (1985), Nisbet and Shucksmith (1986) and Kafai (1993). Since there are a number of common elements between Kafai's study and this study, some comparisons are drawn between the two.

1.4.2 *Introduction to the research design*

As indicated earlier, the research design uses a case study of one general/credit class (twelve girls, eight boys) which extends over eighteen months while they are learning to program. The justification for adopting a case study and for the methods of collecting and evaluating the data is presented in chapter 4, together with an analysis of their strengths and limitations. The methods of conducting collaborative action research are discussed fully in chapter 3 in relation to the programming project.

A key feature of the case study is that it involves an *intervention* since it is concerned with the development and evaluation of an effective learning environment for pupils. I could, instead, have observed, measured and analysed the existing learning, teaching and assessment processes across a range of programming classes without seeking to intervene.

Scriven (1980) contrasts descriptive approaches (which aim to describe a method) with prescriptive approaches (which aim to prescribe a method), and argues that it is a mistake to assume that the first is always a necessary or sufficient step towards the second. This argument is illustrated well by an example drawn from his own experience:

> I have recently had occasion to run an experiment that involved collecting 6000 scripts from high-school students in California, who were asked to deal with a simple problem in practical logic. It would be a staggering waste of time to try to develop a model that accounts for the

> resulting mess in any detail ... We should instead turn to the much more important task of trying to improve this performance. We don't need to understand the minute mechanisms that generate the profusion of errors in order to do this – these mechanisms aren't likely to be of interest, just *because* the results are so bad. *Broad* patterns may be helpful in order to suggest ways to improve. The search for models of error production becomes more worthwhile as we come nearer to closing the gap between good performance and actual performance. It is a fatuous search when a dozen plausible strategies for badly needed improvement lie ready to be tested, and a million "random" errors are being made. The fact that *sometimes* the errors contain clues for remedies in no way supports the substitution of the task of finding a model for them, for the task of finding a way to avoid them. (p.129)

From this account it makes sense to situate the case study within the programming project because, through the project, we had already begun to close the gap between good and actual performance. The existence of a set of 'plausible strategies for badly needed improvement' is clearly evidenced in the research literature on developing generalizable problem solving, thinking and learning strategies (see, for example, the earlier reference in 1.3.2 to Nisbet and Shucksmith's work).

There is also a moral purpose to teaching, which Fullan (1993, p.111) perceives as, 'making a difference in the lives of more and more individual students.' He argues that this moral purpose needs to be highlighted in teacher education and teaching, and re-conceptualized as a theme for change.

There are issues of evaluation implicit in Scriven's account:

> ... evaluation is part of all problem solving, although it's the hidden part, it may be like the hidden part of an iceberg. Even to recognize that a problem exists presupposes the capacity for a negative evaluation of the immediate responses. And to recognize a solution when restructuring or algorithm application turns one up is to evaluate a putative solution favorably. Manufacturing without quality control is not manufacturing, it's garbage production; and inventiveness without evaluation is not problem solving, it's free association. (Scriven, 1980, p.132)

Therefore, if one construes this thesis as, in part, an exercise in problem solving, guided by moral purposes, it must possess a strong evaluative focus throughout. However the judgements that this evaluation calls for are non-trivial. Nisbet and Shucksmith (1984, p.16) recognized this in their research with teachers – to create the kind of learning situations in classrooms that would encourage learning to learn (their 'seventh sense') – when they described it as 'exploratory work', 'speculative', but nevertheless of potential value:

> The seventh sense is, admittedly, in a different category of research from grade-related criteria in secondary school examinations; but it may prove its relevance in time when grade-related criteria are dead and gone.

In an article which discusses the contribution of research to education, Nisbet (1988, p.18) argues that the definition of relevance needs to be drawn widely, 'to ensure that important issues are not excluded from the agenda, to give alternative viewpoints fair consideration, and even to awaken new expectations of what might be possible.' In controversial areas, Nisbet sees the influence of research on educational policy making and practice as, 'indirect and longer-term, through analysis, new interpretations and new concepts accumulating over time to influence the climate of opinion – not so much offering solutions to problems but rather defining the problem to which solutions must be sought...' (ibid., p.15)

However seeking to define a problem clearly is not incompatible with a solution oriented research strategy, so long as the problem is not too narrowly construed. Deep engagement with a problem should enable it to be seen in a clearer light. This thesis is therefore about 'defining the problem' as well as about exploring solutions.

1.5 THESIS CHAPTER OVERVIEW

The thesis is divided into nine chapters.

In **chapter 2,** I explore the main themes and issues that the research is designed to address. Through the research literature on teaching thinking and problem solving I explore the following themes: embedding higher order thinking and problem solving into academic disciplines, teaching for understanding, teaching for transfer, learning to learn, and the affective and social dimensions of learning and thinking. I also explore a range of issues surrounding the teaching of programming in schools – its justification, transfer from programming, and methods of teaching. Issues of assessment and evaluation are discussed in context throughout the chapter.

In **chapter 3** the development of the programming project and its outputs are examined. The manner in which the project engaged teachers in collaborative action research to create an effective learning environment for pupils is the focus of the first part of the chapter. The nature of the outputs (the learning resources) and the design principles on which they are based provide the focus for the second part of the chapter (some theory is introduced on resource-based teaching and differentiation).

In **chapter 4** the case study design, its conduct and evaluation are presented and discussed. The following aspects are covered: the criteria for selecting the case study class; the case study setting; how consent was gained from pupils; key features of the design of the learning environment; the range of data collection methods utilized; methods of evaluation; justification for using a case study design and for the methods of collecting and evaluating data; and the strengths and limitations of the research design.

The next four chapters present and discuss the case study findings, as follows: **chapter 5** – pupils' progress in learning, gleaned from records of progress and the folder of work; **chapter 6** – pupils' attainments, gleaned from assessment records and the assessment portfolio; **chapter 7** – pupils' affective responses to learning to program, gleaned from questionnaires and interviews; and **chapter 8** – pupils' reflections on learning and problem solving processes, gleaned from review tasks and interviews, and evidence on near transfer. Throughout the discussion of findings, evidence on the performance of the learning environment against a range of evaluation criteria (stated in chapter 4) is accumulated.

Chapter 9 is the final chapter in which the main conclusions and implications of the research are presented.

In addition to the bibliography and appendices, there is a glossary of computing and programming terms. The learning resources and full interview transcripts are not appended because of their length, but they can be obtained from the author.

2 KEY THEMES AND ISSUES

2.1 CHAPTER OVERVIEW

This chapter examines through the literature the main themes and issues that are relevant to the research. These can be classified as relating to broader educational goals – the development of thinking skills, problem solving, metacognition and understanding – and issues surrounding the teaching of computer programming in schools.

The sections of this chapter are as follows: 'the thinking curriculum' – on approaches which seek to embed higher order thinking and problem solving skills into academic disciplines (section 2.2); teaching for understanding (section 2.3); teaching for transfer (section 2.4); 'learning to learn' – on approaches which seek to develop learning strategies in the real context of learning (section 2.5); the affective and social dimensions of learning and thinking (section 2.6); and issues surrounding the teaching of computer programming in schools (section 2.7). The issues of assessment and evaluation are discussed throughout the chapter.

All of the themes and issues are explored in later chapters. The literature on research methods is introduced mainly through the discussion on collaborative action research (in chapter 3) and the discussion on the design, conduct and evaluation of the case study (in chapter 4).

2.2 THE 'THINKING CURRICULUM'

This section examines ideas on embedding higher order thinking and problem solving into academic disciplines (infusion). It addresses the following questions: (1) what is the justification for a 'thinking curriculum'? (2) what are appropriate definitions for higher order thinking and problem solving in academic disciplines?; (3) what are the main arguments for and against separate thinking skills programmes versus infusion?; (4) what are the main assessment and evaluation issues? and (5) which teaching methods are likely to be effective for infusion?

2.2.1 Justification

Why is it important to try to improve the quality of student thinking? Swartz and Parks (1994, p.3) observe of the current educational scene in the USA:

> Improving the quality of student thinking is an explicit priority of current educational reform efforts. Recommendations from groups ranging from

> education commissions to the nation's governors support this priority and affirm that good thinking is essential in meeting the challenge of living in a technologically oriented, multicultural world.
>
> While these recommendations have been advanced primarily because of the projected demands of the work force in the twenty-first century, they also reflect an awareness that knowledgeable thinkers have a better chance of taking charge of their lives and achieving personal advancement and fulfillment. Our students must be prepared to exercise critical judgment and creative thinking to gather, evaluate, and use information for effective problem solving and decision making in their jobs, professions, and in their lives.

Although the importance of improving the quality of student thinking is recognized, the path to achieving this goal is far from clear, according to Resnick (1987, p.1):

> The question of whether schools can do a better job of teaching American children "higher order skills" is very much in the air. ... Yet beyond the agreement that our schools ought to be doing better than they are at building the intellectual capabilities of American young people, it is extremely difficult to discern what really should and can be done.

In Scotland much the same kind of concerns have arisen. These are expressed in curricular documents such as those reviewed by Holroyd (1989) when he examined the place of problem solving in the secondary curriculum, and more recently in the Higher Still reforms. Also there are many pronouncements of a general kind in official publications on the need for a better qualified workforce which educational and training establishments are called upon to provide, in order to, for example, '... meet the international challenges of competitiveness, innovation and quality,' and to respond to, '... rapid technological and other changes throughout our society.' (Scottish Office, 1997, p.1) It is recognized nowadays that basic literacy and numeracy, although necessary, are insufficient to meet these demands.

Nisbet and Davies (1990) list recent curriculum reforms in countries throughout the world – Canada, Finland, France, Italy, Japan, Norway, the UK, North America – which are aimed at developing the abilities to think and solve problems, to learn and to understand.

2.2.2 *Definitions*

'Higher order thinking is difficult to define but easy to recognize when it occurs.' Thus Resnick (1987, p.44) begins the summary and conclusions of *Education and Learning to Think*, a paper which sets out to address what

American schools can do to more effectively teach what have come to be called 'higher order skills'. The paper begins by addressing the problem of definition:

> Philosophers promote critical thinking and logical reasoning skills, developmental psychologists point to metacognition, and cognitive scientists study cognitive strategies and heuristics. Educators advocate training in study skills and problem solving. How should we make sense of these many labels? Do critical thinking, metacognition, cognitive strategies, and study skills refer to the same kinds of capabilities?
>
> (ibid., p.1)

Resnick provides a working definition of higher order thinking (ibid., p.3):

- Higher order thinking is *nonalgorithmic*. That is, the path of action is not fully specified in advance.
- Higher order thinking tends to be *complex*. The total path is not "visible" (mentally speaking) from any single vantage point.
- Higher order thinking often yields *multiple solutions*, each with costs and benefits, rather than unique solutions.
- Higher order thinking involves *nuanced judgment* and interpretation.
- Higher order thinking involves the application of *multiple criteria*, which sometimes conflict with one another.
- Higher order thinking often involves *uncertainty*. Not everything that bears on the task at hand is known.
- Higher order thinking involves *self-regulation* of the thinking process. We do not recognize higher order thinking in an individual when someone else "calls the plays" at every step.
- Higher order thinking involves *imposing meaning*, finding structure in apparent disorder.
- Higher order thinking is *effortful*. There is considerable mental work in the kinds of elaborations and judgments required.

There are several corollaries that stem from this definition:

- Any teaching programme that seeks to develop higher order thinking must also be concerned with developing *problem solving skills* (since higher order thinking is nonalgorithmic, complex, often yields multiple solutions, involves nuanced judgement, the application of multiple criteria, and uncertainty);

- Any teaching programme that seeks to develop higher order thinking must also be concerned with developing *metacognition* (since self-regulation of thinking requires effective self-monitoring);

- Any teaching programme that seeks to develop higher order thinking must also be concerned with developing *understanding* (since imposing

meaning and finding structure are ways of getting to understand complex situations at a deep level);

- Any teaching programme that seeks to develop higher order thinking must also be concerned with developing *motivation* (since higher order thinking is effortful).

This working definition of higher order thinking therefore serves to highlight the connectedness of the key themes that are explored in the research. It provides the definition of higher order thinking that is adopted in this research. Within this definition, problem solving can be viewed as a *goal oriented process which requires the integrated use of a range of higher order thinking skills*, such as generating ideas, making interpretations and judgements, and using strategies to manage the complexity of situations.

Resnick's definition of higher order thinking differs markedly from Bloom's influential taxonomy of educational objectives where problem solving and other thinking activities are placed at the top of the hierarchy. Resnick and Klopfer (1989, p.3) argue that such theories had the effect of isolating thinking and problem solving from the main activities of learning: 'Thinking and reasoning became not the heart of education but hoped-for capstones that many students never reached.' However recent cognitive research points towards higher order thinking being the hallmark of successful learning at all levels:

> Children cannot understand what they read without making inferences and using information that goes beyond what is written in the text. They cannot become good writers without engaging in complex problem-solving-like processes. Basic mathematics will not be effectively learned if children only try to memorize rules for manipulating written numerical symbols. All of this implies that "basic" and "higher order" skills cannot be clearly separated. (Resnick, 1987, p.45)

Resnick takes this argument further by suggesting that the term 'higher order' skills is fundamentally misleading because it suggests that another set of skills, presumably called 'lower order' skills, has to come first.

> This assumption – that there is a sequence from lower level activities that do not require much independent thinking or judgment to higher level ones that do – colors much educational theory and practice. Implicitly at least, it justifies long years of drill on the "basics" before thinking and problem solving are demanded. Cognitive research on the nature of basic skills such as reading and mathematics provides a fundamental challenge to this assumption. Indeed, research suggests that failure to cultivate aspects of thinking such as those listed in our working definition of

> higher order skills may be the source of major learning difficulties even in elementary school. (Resnick, 1987, p.8)

These ideas form the basis of the 'Thinking Curriculum':

> The Thinking Curriculum calls for a recognition that all real learning involves thinking, and thinking ability can be nurtured and cultivated in everyone, and that the entire educational program must be reconceived and revitalized so that thinking pervades students' lives from kindergarten onward ... (Resnick and Klopfer, 1989, p.2)

Underpinning the 'Thinking Curriculum' is a constructivist view of learning:

> Cognitive scientists today share with Paigetians a constructivist view of learning asserting that people are not recorders of information but builders of knowledge structures. To know something is not just to have received information but also to have interpreted it and related it to other knowledge. To be skilled is not just to know how to perform some action but also to know when to perform it and to adapt the performance to varied circumstances. (Resnick & Klopfer, 1989, p.3–4)

An important theme is using knowledge *generatively*. Key concepts and principles have to be called upon repeatedly as ways to link, interpret and explain new information. Therefore concepts and principles are continually at work in contexts of reasoning and problem solving. A central problem for educators is therefore how to help students get started in developing their base of generative knowledge so they can learn easily and independently later on.

The thinking curriculum must also attend to developing students' *motivation* to use the knowledge and skills they have learned. The social setting may help to shape students' disposition to engage in thinking through letting students know that all elements of critical thought – interpretation, questioning, trying possibilities, demanding rational justifications – are socially valued (Resnick and Klopfer, 1989).

According to Nickerson (1987) conventional teaching is unlikely to achieve the goal of enabling students to acquire good thinking skills. This goal is not likely to be realized spontaneously or as an incidental consequence of attempts to accomplish other goals. Therefore explicit attention must be paid towards it (this theme is explored more fully in later parts of this chapter).

2.2.3 *What are the arguments for and against infusion?*

Transfer emerges as a key issue in the arguments for and against infusion, and therefore it seems appropriate, before presenting the arguments, to provide an acceptable definition of the term. How does transfer go beyond ordinary

learning? Perkins and Salomon (1988) answer this question by saying that *the skill or knowledge in question has to travel to a new context – across a gap that might in principle block it.* They recognize this as a fuzzy distinction, but say that in practice we have a rough sense of what gaps might be significant. This distinction between transfer and ordinary learning provides a working definition that fits well with the way in which other researchers discuss transfer – generally without defining their use of the term.

The main division that Nisbet (1990) identifies in the field of teaching thinking is between those who aim to teach thinking skills through specially designed *programmes*, and those who favour the *infusion* of thinking throughout the established curriculum. Programmes are usually based on analysis of component skills in thinking, which are taught and practised on special courses. The main criticisms of this approach are that it treats thinking as an 'add-on' element, the skills approach is 'reductionist', and transfer of these skills to new contexts is not guaranteed. Those who favour infusion argue that thinking cannot and should not be separated from its context, and transfer is more likely if thinking is embedded in all teaching and learning. However if problem-solving is only seen as a better method of teaching a subject then there is little transfer.

Resnick (1987) reviews a range of current programmes for teaching higher order thinking skills, most of which represent efforts to improve thinking skills through the addition of special courses or course units. She categorizes them broadly according to their focus: problem solving in the disciplines, general problem-solving skills, reading and study strategies, self-monitoring skills, components of intelligence, and informal logic and critical thinking. In most cases, the evidence amounts mainly to data showing that students who have taken particular courses are more likely to perform well on the tasks directly taught in the courses than other students (for example, there is improved problem-solving or laboratory performance in maths or science). Only a few studies have assessed the ability to generalize to other parts of the school curriculum or to out-of-school performance. General improvements in problem-solving or general thinking abilities have rarely been demonstrated, perhaps because few evaluators have included convincing assessments of these abilities. Resnick concludes, although the available evidence does not establish that such courses can produce broad transfer of learning, neither does it allow the rejection of separate courses as an element in an educational reform designed to improve higher order abilities. In this view, a prudent approach would be to seek to embed higher order thinking skills into all the traditional

school disciplines, regardless of what is being done by way of special courses in thinking or learning skills.

Resnick (1987) lists several advantages of a discipline-embedded approach. Firstly it provides a natural knowledge base and environment in which to practice and develop higher order skills ('One cannot reason in the abstract; one must reason about something,' p.36) and as knowledge in a discipline develops, the base on which effective problem solving can operate is established. Secondly, it provides criteria for what constitutes good thinking and reasoning within each discipline. Each style of reasoning is worth learning. Finally, it ensures that something worthwhile will have been learned even if wide transfer proves unattainable. This latter point is given considerable emphasis:

> It amounts to saying that no special, separate brief for teaching higher order skills need be made. Rather, it proposes that if a subject matter is worth teaching in school it is worth teaching at a high level – to everyone. (ibid., p.36)

However this is not proposed as an easy path. Resnick points to the pendulum swings of attention either to process skills ('doing science' etc.) or to building large bodies of knowledge, as being unhelpful to the process of embedding thinking skills into academic disciplines. She concludes that research and experimentation into how to truly combine these aspects are badly needed.

Some of the potential problems of infusion are referred to earlier (in 1.2.6) in relation to teaching problem solving within Standard Grade courses (Holroyd, 1989; Drever, 1988), specifically: problem solving activities are hidden under different titles; problem solving as a useful end in itself may get 'squeezed out' by other priorities; pupils may get confusing messages about problem solving from different subjects; the evaluation of permeating aspects is notoriously difficult; and there is a need for a more precise formulation of the skills that teachers will need to develop pupils' problem solving skills.

2.2.4 Issues of assessment and evaluation

Resnick (1987) discusses assessment and evaluation issues in relation to the range of current thinking skills programmes that she reviewed. Different issues arose within each category of programme (e.g. programmes on problem solving within disciplines tend to be less concerned with transfer than separate programmes on general problem solving skills), but nevertheless there are a number of common issues:

- Current testing practices in American education

Many standardized tests are not adequately sensitive to detect the effects of thinking skills interventions, and their widespread use makes it difficult for such reforms to survive. Even course grades generally provide only an indirect measure and offer no indications of transfer beyond academic settings. Short answer or multiple choice formats permit specific components of reasoning or thinking to be assessed but cannot assess the kinds of integrated thinking that we call 'higher order'. Useful evaluations of programmes require that the *educational outcomes of interest are directly assessed.*

- Formal evaluation of programmes is rarely carried out

Less formally conducted evaluations may report student evaluations which indicate that they feel better about their thinking and learning abilities after the course, describe examples of good problem solving or thinking demonstrated by some individuals, and point to the long-term use of the programme to attest to its popularity. They may include self-reports by students of having used the abilities in question beyond the confines of the programme, but there is no direct evidence that transfer has occurred.

- It can be difficult to unpick the effects of particular interventions from other effects

Sometimes a strategy, skill or teaching method is embedded within a fairly extensive programme. For example, reading comprehension strategies (such as skimming, using context to figure out words and meaning, self-testing, summarizing) may be taught within an overall programme designed to help students to manage their time, control anxiety and mood, and apply deliberate learning strategies. Resnick comments favourably on one evaluator's strategy to overcome this difficulty:

> This mixture of global evaluation with detailed analyses of the effects of specific component strategies, pursued in a cumulative fashion and extended so that long-term effects and transfer can be evaluated, is precisely what we need to establish which elements of complex programs are important to their overall effects. (ibid., p.25)

- It can be difficult to understand the effects of particular interventions

The effects of an intervention cannot always be explained since not everything may be known about the underlying learning mechanisms. This makes it difficult to determine in advance the essential components of a training approach.

Baron (1987) lists four dimensions to aid the planning of a carefully controlled, broadly conceived, and well-documented evaluation study of thinking skills in the classroom: formative/summative, product/process, qualitative/quantitative, and experimental/quasi-experimental. No matter the scale, a sound evaluation should represent a concerted attempt to understand both whether there were any meaningful changes in the students' thinking skills and dispositions, and how these changes came about. She recommends to teacher evaluators that they should: develop jointly with their students a set of criteria for effective thinking to be applied in the classroom; use a wide-angle and telephoto lens; evaluate continually; look for sustained effects; look for transfer; look for side-effects; look for metacognition; use a variety of approaches – discussions, writing, tests, performance tasks, unobtrusive measures which do not interfere with students' learning and do not require additional instruction time; and interpret results. She recommends to programme developers that they should also consider the reliability, validity and sensitivity of available test measures and whether the information deriving from such measures is in a useful form for programme improvement, and they should attempt to evaluate appropriate attitudes and dispositions.

Baron focuses on what should be done, but it is clear (from Resnick's, and other, analyses) that much of this is not being done in current evaluation studies. Given the comprehensive nature of the advice, the relatively uncharted territory that it encompasses and the exigencies of teaching and research, perhaps this is hardly surprising. Nisbet and Davies (1990, p.60) point to some of the general deficiencies and problems:

> The evaluations that have been reported tend to be small-scale, involving disappointingly small numbers or using only a limited measure of effectiveness. Inevitably, evaluations are short-term, since there is insufficient time to follow up any group for long-term effects. It is also difficult to decide what should be taken as firm evidence of success, or to distinguish between the merits of the materials and the methods adopted in using them, or to take account of the quality of the teaching.

In this area of research, trying to get beneath the surface to find out what is 'running through the child's mind as he or she wends his or her way through the task' (Flavell, 1976, p.234) presents a particular challenge. The picture is now becoming less bleak as new trends emerge in assessment practices, as Broadfoot (1994, p.90) outlines:

> In this country [the UK] as in many other parts of the world, we are witnessing the emergence of a new assessment paradigm in which *it is*

> *learning itself,* rather than simply the *measurement* of that learning, which is its central purpose. The work associated with trying to implement records of achievement in various educational contexts has led down a tortuous path of discovery which has included a realisation of the importance of clearly specifying curriculum goals; of sharing them with students; of allowing students to set their own learning targets and of drawing up more general 'action plans'; of the need for students to be equipped with the skills and responsibility for assessing their own work; of the need for teachers and students to review progress together and for students to monitor their own learning strategies.

Simpson (1997, p.98–9) sets out the principles of a partnership between student and teacher in which the student internalizes standards and plays an active role in assessing his or her performance. This example, related by an English teacher, is about a system of profiling:

> We have the assessment criteria set out down one side of the profile, with self-assessment built in, and we are trying to say, 'This is what we expect out of this piece of work, this is what *you* need to do for a good piece of work', in this type of writing, say personal writing, and get them to evaluate their own writing against these criteria and say, 'OK, I'm weak at this' and build in support with worksheets etc. and there's space on the profile for targeting, 'What I need to do next to improve my writing.'

Issues of assessment and evaluation are discussed in detail in later chapters of the thesis.

2.2.5 *An example of problem solving within the disciplines*

Which teaching methods are likely to be effective for infusion? I have chosen to examine here (and in 2.2.6) two contrasting approaches to infusion in order to answer this question. Both are very well thought out and worked through into practice. The first is in the realm of mathematical problem solving – based upon Polya's (1948) ideas on teaching heuristic strategies[4] – which fits within Resnick's first category of *problem solving within the disciplines,* and is not much concerned with fostering transfer beyond mathematics. The second approach seeks to infuse the teaching of critical and creative thinking into content instruction across all aspects of the curriculum, and pays explicit attention to transfer. This approach, which also encompasses problem solving, has been developed by Swartz and Parks (1994).

---

4 Polya defines these as mental operations typically useful in solving problems.

2.2.5.1 Polya's ideas on teaching mathematical problem solving

Polya (1948, p.[v]) provides an elegant statement of effective teaching methods for problem solving within mathematics which has not lost its currency over time. He introduces his ideas as follows:

> A great discovery solves a great problem but there is a grain of discovery in the solution of any problem. Your problem may be modest; but if it challenges your curiosity and brings into play your inventive faculties, and if you solve it by your own means, you may experience the tension and enjoy the triumph of discovery. Such experiences at a susceptible age may create a taste for mental work and leave their imprint on mind and character for a lifetime.
>
> Thus, a teacher of mathematics has a great opportunity. If he fills his allotted time with drilling his students in routine operations he kills their interest, hampers their intellectual development, and misuses his opportunity. But if he challenges the curiosity of his students by setting them problems proportionate to their knowledge, and helps them to solve their problems with stimulating questions, he may give them a taste for, and some means of, independent thinking.

This is clearly not a narrow focus on mathematical skills acquisition, and Resnick and Klopfer's working definition of higher order thinking is much in keeping with its sentiments. Polya links together the affective and cognitive dimensions of problem solving when he refers to 'taste' (the affective dimension) and 'means' (the cognitive dimension).

Polya provides mathematics educators and learners with a framework for tackling problems which splits into four phases: first you have to *understand* the problem, second you have to *devise* a plan, third you have to *carry out* your plan, and fourth, you have to *examine* the solution obtained. Polya's framework is adopted within this research since it provides a close fit to the practical abilities assessable element (see 1.2.4) which breaks down into analysis of problems and design of solutions, implementation of practical solutions, and evaluation of practical solutions.

He explains the importance of each phase, as follows:

> It may happen that a student hits upon an exceptionally bright idea and jumping all preparations blurts out the solution. Such lucky ideas, of course, are most desirable, but something very undesirable and unfortunate may result if the student leaves out any of the four phases without having a good idea. The worst may happen if the student embarks upon computations or constructions without having *understood* the problem. It is generally useless to carry out details without having seen the main connection, or having made a sort of *plan*. Many mistakes

> can be avoided if, carrying out his plan, the student *checks each step*. Some of the best efforts may be lost if the student fails to reexamine and to *reconsider* the completed solution. (ibid., p.6)

It is not difficult for teachers of subjects other than mathematics to think of the equivalent of students embarking prematurely on computations or constructions.

Before discussing and illustrating each of the four phases of the model, Polya makes some general points about how students may be enabled to solve mathematical problems, which I shall paraphrase below:

- Giving help and asking questions

  The teacher should help, not too much and not too little, to give the student a reasonable share of the work. If the student is unable to do much the teacher should leave him some illusion of independent work. It is best to help the student *naturally* – by asking a question or indicating a step that could have occurred to the student himself – and also *unobtrusively* – by stating the question or suggestion generally to leave plenty for the student to do.

  Often the same question or step recurs; it may be worthwhile to group questions and suggestions which are typically helpful in discussing problems with students (as Polya has done in relation to mathematical problem solving). If the same question is repeatedly helpful, the student will be induced to ask the question by himself in a similar situation, and, through experiencing success, he will discover the right way of using it.

- Imitation and practice

  The teacher may have two closely related aims in view when addressing to his students a general question or suggestion – to help the student to solve the problem at hand (which adds a little to his ability to solve problems), and to develop the student's ability so that he may solve future problems by himself.

  Solving problems is a practical skill, like swimming. Trying to solve problems, you have to observe and to imitate what other people do when solving problems, and finally, you learn to do problems by doing them. The teacher must therefore instil some interest for problems in the students' minds and give them plenty of opportunities for imitation and practice. When the teacher solves a problem before the class, he should

> dramatize his ideas a little and should put to himself the same questions that he uses when helping the students.

The following discussion, which introduces the first phase, 'understanding the problem', gives a flavour of the manner in which each phase is introduced:

> It is foolish to answer a question that you do not understand. It is sad to work for an end that you do not desire. Such foolish and sad things often happen, in and out of school, but the teacher should try to prevent them from happening in his class. The student should understand the problem. But he should not only understand it, he should also desire its solution. If the student is lacking in understanding or in interest, it is not always his fault; the problem should be well chosen, not too difficult and not too easy, natural and interesting, and some time should be allowed for natural and interesting presentation. (ibid., p.6)

This allows me to amplify my definition of problem solving as a *goal oriented process which requires the integrated use of a range of higher order thinking skills.* The problems should be *well chosen by the teacher – not too difficult and not too easy, natural, interesting and engaging.*

2.2.5.2 *Developing the ideas: Schoenfeld's work on teaching mathematical problem solving*

Polya's ideas on mathematical problem solving have been very influential within mathematics teaching, and also beyond, for example, Dromey's (1982) text on computer problem solving begins with an acknowledgement to Polya's work and it even borrows its title ('How to solve it by computer') from Polya's seminal text.

Schoenfeld (1985; 1989) has researched extensively into the use of heuristic strategies in mathematics teaching (his work is referred to throughout later chapters of the thesis). In an article on teaching mathematical thinking and problem solving, Schoenfeld (1989) presents some simple problems that nevertheless provide a springboard for discussion and amplification of a range of important mathematical ideas, such as establishing subgoals, working forwards, working backwards, assuming you have a solution and determining its properties, exploiting extreme cases, and so on, all of which are in the Polya tradition. He expands on this theme to introduce other critical aspects of instruction, as follows:

- The creation of a supportive learning environment

The atmosphere in which students learn strategies is very important. Therefore the teacher should:[5] help students accept the challenges; build a supportive classroom atmosphere in which they will be prepared to tackle the unfamiliar and not feel too threatened when they become stuck; allow them to pursue their own paths towards a solution and assist them when necessary, without giving the answer away; provide a framework within which they can reflect on the processes involved and thereby learn from experience; and talk to them about the processes involved in doing and using mathematics, so that they build up a vocabulary for thinking and learning about it.

The explicit focus on processes and strategies is designed primarily to enhance students' ability to think mathematically and to learn mathematics, rather than with a view to enabling students to generalize the processes and strategies to non-mathematical contexts.

- Using a problem-based methodology to enable students to develop a mathematical viewpoint

Schoenfeld also stresses the need for students to be able to experience mathematics as a *sense-making activity*, rather than as, 'a set of disconnected and (to them) arbitrary procedures passed on for memorization.' (ibid., p.100) Certain kinds of ritualized classroom procedures result in students using these procedures in a mechanistic fashion, resulting sometimes in answers that don't make sense or don't match reality. (He gives as an example the answer, '31 remainder 13' as the number of buses needed to transport a group of people. Noting that buses don't come with remainders is a 'reality check'.) A problem-based methodology enables students to experience mathematics as, 'a discipline of reason where mathematical conventions and terminology make sense because they do work for you.' (ibid., p.100)

- Fostering metacognition to encourage self-regulation

Metacognition is the awareness or control of one's own thinking processes (Nisbet, 1990). Schoenfeld asserts that metacognitive skills can be learned as a result of explicit instruction that focuses on metacognitive aspects of mathematical thinking. These aspects are important to select and pursue the right approaches, recover from inappropriate choices, and in general monitor and over-see the entire problem-solving process. 'In broadest terms, this is the issue of control' (Schoenfeld, 1985, p.98–9).

---

5 Here Schoenfeld quotes from Stacey, K. and Groves, S. (1985), *Strategies for Problem Solving*, Burwood, Victoria (Australia): VICTRACC Ltd.

The form of instruction that he recommends is coaching, using active interventions as students work on problems. He describes one obtrusive but effective technique to encourage students to self-regulate[6], in which the teacher reserves the right to ask the following three questions at any time:

> What (exactly) are you doing?
> (Can you describe it to me precisely?)
> Why are you doing it?
> (How does it fit into the solution?)
> How does it help you?
> (What will you do with the outcome when you obtain it?)
> (Schoenfeld, 1989, p.98)

Students become better and better at answering these questions as the term progresses: 'When the students realize that the questions will continue, they begin to defend themselves by discussing the answers in advance.' (ibid., p.98) By the end of the term, discussing the questions has become habitual. The results of these interventions are reflected in Schoenfeld's observations on how one pair of students succeeded in solving an unfamiliar problem:

> by virtue of good self-regulation, the students gave themselves the opportunity to solve the problem. They curtailed one possible wild-goose chase after beginning to work on the problem, and truncated extensive computations half-way through the solution. (ibid., p.98–9)

Since the issues seem to be the same across disciplines – whilst engaged in a complex task you have evaluated the current state of affairs and decided to do something about it – Schoenfeld highlights the potential for self-regulated thinking to transfer to other contexts, but he does not take this idea further to propose methods of instruction to foster transfer to other contexts.

Polya also does not ignore the potential for transfer of strategies beyond mathematics. When he stresses the need for generality in the questions and suggestions, he observes: '... we can ask them with good effect dealing with all sorts of problems.' (Polya, 1948, p.2) When he stresses the need to pose questions naturally, he draws parallels to common-place situations. He does not, however, under-estimate the role of local knowledge in successful problem solving, which may prevent transfer of strategies from occurring:

> We know, of course, that it is hard to have a good idea if we have little knowledge of the subject, and impossible to have it if we have no

---

6 This technique is not at odds with Polya's recommendation that any questioning should be *unobtrusive*, since Polya uses this term to mean that any questions should be stated generally to leave the student plenty to do.

> knowledge. Good ideas are based on past experience and formerly acquired knowledge. Mere remembering is not enough for a good idea, but we cannot have any good idea without recollecting some pertinent facts... (ibid., p.8)

The central issue of the inter-relationship between general strategic knowledge and specialized domain knowledge is returned to later in this chapter when discussing teaching for transfer.

2.2.6 *An example of infusing thinking skills across the curriculum*

2.2.6.1 *Swartz and Parks' infusion approach*

Swartz and Parks (1994) have produced a lesson design handbook for elementary grade teachers on infusing the teaching of critical and creative thinking into content instruction. They explain infusion as blending explicit instruction in thinking skills and processes with content instruction, using methods which enhance students' thinking and comprehension of the content.

They also explain how infusion is similar to and different from separate thinking skills programmes (which they describe as 'teaching of thinking') and methods of teaching to promote students' deep understanding of the content (which they describe as 'teaching for thinking'). In both 'teaching of thinking' and infusion students learn how to use explicit thinking strategies, but with infusion there is no need to 'bridge' into the curriculum. In both 'teaching for thinking' and infusion, methods to promote deep understanding are used (e.g. higher order questioning, Socratic dialogue, co-operative learning) but infusion lessons are also characterized by direct instruction in thinking skills and processes. This difference is illustrated by an example:

> "Why did the plague spread so rapidly in medieval Europe?" is a challenging question and unlike the question "What were the dates of the plague in medieval Europe?" provides an opportunity for higher order thinking.
>
> This kind of questioning, however, remains content oriented. Its goal is usually to yield a deeper understanding of what is being taught. When students respond by mentioning factors like lack of sanitation or lack of medical knowledge, teachers usually ask students to elaborate those responses so that the class can develop a rich understanding of conditions that could cause such a devastating epidemic. The product (student answers), rather than the process (student thinking), is the focus in these lessons.

> ... *How* students arrive at their responses remains implicit. While some students may respond thoughtfully, others may respond hastily and unsystematically. Some students may not respond at all. (ibid., p.10)

Swartz and Parks argue that teachers must take time to clarify the skilful thinking needed, in order to yield more thoughtful responses from more students. They identify the following three principles, which emerged from the 'thinking skills' movement of the 80's in America, as providing the basic rationale for their approach:

- The more explicit the teaching of thinking is, the greater impact it will have on students.
- The more classroom instruction incorporates an atmosphere of thoughtfulness, the more open students will be to valuing good thinking.
- The more the teaching of thinking is integrated into content instruction, the more students will think about what they are learning. (ibid., p.3)

They argue that infusion is a natural way to structure content lessons:

> The curriculum is not a collection of isolated bits of information but the material that informed, literate people use to make judgments. We expect that information about nutrition should influence students' dietary habits. We expect that an understanding of American political history should affect how citizens vote. We expect that a deep understanding of a character's motivation and actions in a work of fiction should inform a discerning reader about his or her own conduct and responsibility. (ibid., p.3–4)

... and express confidence that all teachers, with guidance, can help all students to become better thinkers (which accords with Resnick and Klopfer's vision of the 'thinking curriculum'):

> Although textbooks and tests are changing to reflect this aim, it is the classroom teacher who, through day-to-day instruction, must assume the main responsibility for helping our students become better thinkers. The effort that is required to meet this goal must, therefore, be directed at effective classroom implementation. (ibid., p.3)

What is meant here by becoming a 'better thinker'? Swartz and Parks stress that students do not need to be taught to do thinking – they do this routinely every day – but rather to *perform thinking tasks skillfully.* This is illustrated by some examples of everyday thinking tasks performed unskillfully which result in hasty and ill-considered decisions. (Nickerson (1987) talks of thinking more effectively – more critically, more coherently, more creatively, more deeply – than we often, perhaps typically, do.)

2.2.6.2 Types of skilful thinking

The types of skilful thinking that they discuss fall into three main categories: skills at generating ideas (creative thinking skills); skills at clarifying ideas (which involve analysis and enhance our understanding and the ability to use information); and skills at assessing the reasonableness of ideas (critical thinking skills which lead to good judgement). Each thinking skill can be taught, reinforced and elaborated in many contexts, and the same content material can be used to teach a different thinking skill. The infusion of key thinking skills into content instruction can be seen to add richness and depth into the curriculum (the teachers' handbook presents sample lesson plans around each skill).

These skills blend together for thoughtful decision making and problem solving: 'We should try to generate original solutions to problems; we should base our decisions on relevant information; and we should assess the reasonableness of each option in order to select the best one.' (ibid., p.6) There is therefore an interplay of creative and critical thinking in the decision making and problem solving process.

The way in which creative and critical thinking come together in authentic tasks is a common theme in the literature, for example, Perkins (in Brandt, 1986, p.15) describes the two as 'hand-in-glove', although he recognizes that one can still talk in terms of balance or emphasis:

> The creative thinker has to be critically aware, because creative thinking, except in the simplest situations, involves the generation and sifting of possibilities and reworking them. That has to be a critical process.

Also Swartz (1987) points to the danger of overly narrow or overly fanciful thinking if the two are separated, and Paul (1987) argues that creativity is essential to all rational dialogical thinking, which is a process in which imagination and its creative powers are continually called forth.

Problem solving has not occurred until a solution has been successfully implemented; we think through and choose the best solution (just as in decision making), however carrying out a solution may pose a new set of problems. One can draw a distinction between the set (or global) problem, and other problems that emerge during planning and implementation (which I refer to in the thesis as 'emergent' problems). The disposition to use skilful thinking when faced with a problem must be developed.

A dual focus on thinking skills and processes is called for (the teachers' handbook presents sample lesson plans around decision making and problem solving processes):

> Teaching the thinking skills of clarification, creative thinking and critical thinking without helping students learn how to use them in decision making and problem solving accomplishes only part of the task of teaching thinking. Teaching strategies for problem solving and decision making, without teaching students the skills needed to use these strategies effectively, is similarly limited.
>
> (Swartz and Parks, 1994, p.8)

Swartz and Parks assert that the problem-solving examples found in mathematics and science texts often provide an inadequate context to teach skilful problem solving, since they are intended to give students practice in applying mathematical or scientific principles rather than grappling with problems that require definition, the selection of the best solution, and implementation. The range of thinking skills that students use in this context is therefore quite restricted.

2.2.6.3 *Instructional methods*

A range of instructional methods is used in infusion lessons for different purposes: to teach the thinking skills and processes, to foster thinking collaboratively, to prompt students to learn content thoughtfully, and to promote thoughtful habits of mind. Each method is fully explicated and exemplified.

- *Direct instruction* is used to promote clarity and reflection about the thinking skill or process.
- *Guided practice* is woven through the lesson with students carrying out their thinking in a *social context.* Individual thinking is blended with group interaction to show students how the interplay of ideas can enhance their thinking.
- Thoughtful learning of the content is prompted through a variety of methods appropriate to the content and students' level of cognitive development (e.g. asking higher order questions, prompting student questioning, 'hands-on' investigation, essay writing).
- Problem-based learning activities challenge students with authentic problems whose solutions require specific content knowledge and conceptual understanding that the teacher helps students to acquire. The

teacher designs an infusion lesson in skilful problem solving around the activity.

- Thoughtful habits of mind are promoted through the teacher *modelling* his or her own thinking dispositions (e.g. by asking clarifying questions, allowing time for students to respond thoughtfully to questions, using precision in language and promoting precision of expression).

An infusion lesson has four distinct phases – *introduction, thinking actively, thinking about thinking,* and *applying the thinking.*

In the *introduction* phase, a discussion or activity demonstrates to students what they already know about the thinking skill or process being taught, shows students why this type of thinking is important and relates its importance to their own experience, and introduces them to the process of engaging in the thinking skillfully and to the significance of doing this in relation to the content they are learning.

In the *thinking actively* phase, verbal prompts (usually questions) and graphic organizers (to reinforce the verbal prompts) are used to guide students through the thinking activity. A 'thinking map' is provided for each thinking skill and process and students may also (or instead) produce their own. For example, the thinking map for skilful problem solving poses the following five questions: Why is there a problem? What is the problem? What are possible solutions? What would happen if you solved the problem in each of these ways? Which was the best solution?

In the *thinking about thinking* phase, there is a 'metacognition map' which provides three organizing questions to guide students' reflections: What kinds of thinking did you engage in? How did you carry out this kind of thinking? Is this an effective way to engage in this kind of thinking? This phase of the lesson serves a retrospective function by bringing to students' consciousness the structure of the thinking they just did, and a prospective function by getting students to develop an explicit written plan to guide their thinking when a similar need arises in future.

Finally, in the *applying the thinking* phase teachers are reminded to use different types of examples to demonstrate the versatility of the thinking skill or process. Students are given additional practice on similar examples to the one they considered in the main lesson activity (near transfer) and on quite different examples (far transfer). Students are asked to guide their own thinking in this phase, using the thinking plan they developed in the third

phase of the lesson. There should be reinforcement of the same thinking throughout the school year.

2.2.6.4 *Contexts and assessment*

To prompt teachers to create their own infusion lessons there is a lesson plan template, and, for each thinking skill and process, a completed lesson plan and list of examples, suggested by teachers, of lesson contexts classified by grade level, subject and topic.

Teachers are advised to take great care when choosing *contexts* for infusion lessons. The key is to find a context that offers rich development of the thinking process *and* rich instruction of the content objective. There are two extremes to be avoided: the content is so complex that it overshadows the thinking activity to the extent that the student remembers the content but not the thinking process; or the content is not well developed (it is superficially touched upon to provide a vehicle for teaching thinking) so that the student learns little about the subject.

The last section in the lesson plan template is on *assessment* of the thinking skill or process. The advice here is that teachers can use the transfer and reinforcement examples to assess the thinking skill or ability, taking note of students' responses or actions. For example, in order to assess problem solving, you can ask students to think through a solution to a problem:

> Make sure the problems you suggest to students are authentic and open-ended and need defining in order to get the most comprehensive profile of your students' problem-solving abilities. Ask students to use a graphic organizer to record their ideas, but let them choose which one they will work with. Also ask them to write down their plan for implementing their solution. Make sure they are raising the questions on the thinking map as they try to solve the problems... (ibid., p.86)

Alternatively, give students a problem solving task to do:

> ask them to plan out their performances. This should provide some information about their problem-solving abilities. As you observe their performance, note how they identify problems, think through possible solutions, and make judgments about the best solution. If you have trouble ascertaining whether they are doing this, you can interview the students after they have completed the task. (ibid., p.86)

2.2.7 *Discussion on the 'thinking curriculum'*

Both of these examples – the first chosen to exemplify problem solving within the disciplines, and the second to exemplify the infusion of thinking skills

across the curriculum – provide rich opportunities to draw out the key themes and issues on the teaching of thinking. Although there is clearly a difference in focus and emphasis between them, there are many overlaps in the teaching methods proposed.

Direct instruction and *modelling* are central features of both teaching methods. Swartz and Parks' suggestion of using thinking maps is analogous to Polya's suggestion on grouping the questions or suggestions that are typically helpful when discussing problems with students. There is, however, a difference in the manner in which the questions are posed. Polya's questions or suggestions are clearly designed to prompt strategic thinking within a mathematical context, as the following set of related questions illustrates: 'What is the unknown? What are the data? Introduce suitable notation. What is the condition, linking a, b, c and x? Is the condition sufficient to determine the unknown?'.[7] This particular set of questions could be stated more generally to be useful beyond mathematics (e.g. 'What is it I need to find out? What information do I already have or could obtain that is relevant to the solution?' etc.), but may be more helpful in a mathematics classroom when posed in the manner that Polya suggests, so long as the student, as well as the teacher, understands them.

Whether a more general embodiment of Polya's questions and suggestions would resemble the 'skilful problem solving' map of Swartz and Parks is a moot point. The latter is designed with open-ended problems in mind, and not all mathematical problems are of this nature. (However Schoenfeld illustrates how many closed mathematical problems can be easily turned around into open-ended problems, e.g. instead of getting students to prove the properties of a two-dimensional shape they could instead be asked to discover them.) Therefore a mathematics teacher might be ill-advised to abandon the use of Polya-like prompts in favour of a more generally conceived plan.

The intention behind both strategies is that students should eventually self-prompt. Getting students to construct their own thinking maps (with teacher guidance) as Parks and Swartz recommend would encourage this by giving students ownership of the plan, helping with retention, and demonstrating to students that they have insight and thinking capability.

Developing metacognition is a key intention behind both teaching methods. A range of teaching strategies is employed to bring this about. Direct instruction

---

7 Polya discusses two types of mathematical problem – 'problems to find' and 'problems to prove'. 'Problems to find' may be theoretical or practical, abstract or concrete, serious problems or mere puzzles. The aim is to find the unknown.

on heuristic strategies or thinking skills and processes, combined with modelling, should heighten students' awareness of their own thinking processes. Thus Polya suggests 'thinking aloud' and dramatizing the solution in front of the class, Schoenfeld suggests coaching students to encourage self-regulation and engaging students in discussion and reflection on processes (Schoenfeld), and Swartz and Parks suggest modelling thinking dispositions and devoting part of the lesson to 'thinking about thinking'.

Problem-based learning activities provide authentic contexts for acquiring content knowledge and conceptual understanding, and also for skilful problem solving to be practised. The *social context* provides a supportive atmosphere in which to learn strategies, opportunities for collaboration, and it may shape the disposition to engage in skilful thinking.

Application of the thinking skill or process to different contexts (far transfer) is a feature only of infusion lessons. Swartz and Parks alert teachers to the need to select the context for the main lesson activity carefully to ensure that content objectives can be met in addition to developing the thinking skill or process in question. If content objectives are not being met, it would seem to be futile to continue to pursue an approach which is intended to enable the thinking skill or process to transfer. In a secondary school, this would amount to the mathematics teacher doing the geography teacher's work and vice versa. Thus there needs to be very careful monitoring to ensure that this does not happen.

The opposite scenario arises when the content objectives swamp the thinking skill or process. Swartz and Parks link this scenario to the content being too complex. There is another important factor to be considered, which is that infusion lessons must be fitted into existing curricula which may not allow adequate *time* to develop thinking skills or processes. When there is competition for time, the aspects which are examined win hands down with most teachers. Therefore the nature of examinations has to change to reflect the new emphasis on thinking skills and processes, and the content sections of syllabuses have to be pared down (however this is never an easy battle to win since it is equated with diluting standards in the minds of many educators).

When selecting a context for infusion lessons it is not always easy to assess the balance between the content and thinking skills objectives. The thinking skill may be so embedded in the content, forming an important part of the epistemology of the subject, that it is difficult to untangle (such is the case for problem solving within programming).

The infusion approach that Swartz and Parks have developed addresses the main criticisms that are made of separate thinking skills programmes: thinking is not treated as an 'add on' element to the curriculum; it demonstrates in realistic contexts the integrated use of thinking skills in decision making and problem solving, and is therefore not reductionist; and teaching for transfer forms part of infusion lessons. It also addresses some of the issues that are raised by Holroyd (1989) and Drever (1988) concerning the need for greater visibility of problem solving across the curriculum through the consistent use of language, consistent treatment across different subjects, and a clear and detailed formulation of the skills that teachers will need to design infusion lessons on problem solving. However if teaching on strategies (e.g. using a thinking map to guide decision making) is done in a mechanistic fashion the benefits of the approach could be lost. There is a real danger that this could occur if the method was adopted as a 'whole school' approach, without adequate staff development or seeking to convince teachers of its value.

Polya's four phases of problem solving and the associated heuristics are derived from a detailed introspective examination of the epistemology and pedagogy of mathematics. If attention were given to generalizing the strategies beyond mathematics this would create the circumstances to examine the extent of overlap between mathematical problem solving and other 'types' of problem solving. When the extent of overlap is known, it becomes easier for teachers (and particularly secondary teachers) to bridge to other contexts. This approach avoids the skewing that could result from attempts to apply some idealized model of problem solving across a range of subjects, and the possible substitution of 'weak methods' in place of 'strong methods' within these subjects.

It should be possible to accommodate both approaches to the infusion of thinking skills and processes into curricula. Many of the infusion lesson examples that Swartz and Parks provide show how naturally the approach can be made to work *in some contexts* (and I can see how it could fit easily into some computing lessons), but if it is pushed beyond its limits it could undermine students' understanding of the content and alienate teachers.

The successful approaches which Schoenfeld outlines in relation to the teaching of mathematical problem solving have their counterparts in other disciplines, e.g. in the studies which sit alongside Schoenfeld's in Resnick and Klopfer (1989). Within the secondary school, there is great untapped potential for cross-fertilization of ideas on developing students' thinking skills and

processes, particularly for aspects such as reading comprehension which cut across subject matters. Resnick (1987) views this as a particularly promising approach to improving thinking skills across the curriculum. Nisbet and Davies (1990) argue that if the aim of education is to produce good thinkers and not just good lawyers or scientists or historians, then there must be some coherent policy of 'thinking across the curriculum'.

2.3 TEACHING FOR UNDERSTANDING

One of the key themes to emerge in the previous section is the importance of fostering a deep understanding of subject matter. This theme will be explored to examine the main pedagogical issues and the links to other themes.

2.3.1 What does it mean to 'teach for understanding'?

When Resnick presents the case for using a discipline-embedded approach to developing higher order thinking *(see 2.2.3 – arguments for and against infusion)* she sees as an advantage of this approach that something worthwhile will have been learned even if transfer proves unattainable, which, she argues, amounts to saying that if a subject matter is worth teaching it is worth teaching to everyone at a high level. This implies the need to foster in all students a deep understanding of the subject matter.

Lampert (in Brandt, 1994, p.28) equates understanding with sense-making:

> It means the conversation in the classroom is dominated by the questions, 'Does this make sense to you?' and 'Why or why not?'
> And it involves communication – being able to explain your thinking so that someone else grasps your ideas. So teaching for understanding has two components: the individual component and the shared component. From that standpoint, I see my responsibility as a teacher as both helping individuals to understand and helping them to express their understanding so that other people in the group understand.

Lampert suggests that, to the typical pattern of exchange between teacher and student – question, answer, teacher's judgement – needs to be inserted the question, 'Why do you think that?', because ...

> when a student gives an answer, he or she has only begun to participate in the conversation. The teacher needs to work to understand the student's thinking before passing judgment on the answer. (ibid., p.28)

The expectation that people should do things and say things that make sense applies across different subject matters, but what counts as an explanation differs from subject to subject: 'Mathematical evidence is different from the

evidence one would use to support a claim in history or literary criticism or economics.' (ibid., p.28–9) Therefore understanding has to be developed subject by subject.

It is important to establish a culture of respect for other people's ideas and for your own. Lampert provides an example from her own teaching, when she asked fifth graders a maths problem on rate and ratio: if a car is going at a constant speed of fifty miles per hour, how far will it go in ten minutes? Rather than saying 'that's wrong' or 'that's ridiculous' when she gets responses like five miles or five hundred miles, she says, 'Does that make sense?' and 'Let's think about it,' which are cues for discussion with the class and for homing in on a solution. She summarizes the strategy as follows:

> Well, first of all, I asked questions that would lead them to question their assumptions. I monitored the discussion so that students could challenge one another in ways that were civil and relatively safe. And when their thinking came close to a big mathematical idea, I helped them to see those connections. (ibid., p.29)

Active, engaged, constructivist learning lies at the heart of this approach. Lampert asserts that there is no simple formula for 'teaching for understanding' that can lend security to student teachers or teachers. Research might come up with such a formula under very carefully controlled conditions, but 'every classroom is different, every child is different and every moment of teaching is different.' (ibid., p.30) For this reason, teachers need more time and opportunity to work together on the problems of their practice and to develop a mutual understanding about what it is important to learn and how to teach it. However the way in which teachers are treated prevents this from happening:

> They've not been treated as people who can and should think about curriculum and instruction. They've been treated as people who have to be told what to do, who can't think for themselves. And that seems particularly ironic when we hear so much about trying to get *children* to think for themselves. (ibid., p.30)

The concept of 'teacher as learner' is explored in the next chapter in relation to the programming project.

2.3.2 *Project Zero: a framework for teaching for understanding*

Project Zero is a recent (1990–5) Harvard University research project which has as its theme enhancing disciplinary understanding in teachers and students. The project investigators were Howard Gardner, David Perkins, and Vito

Perrone, and the project manager Rebecca Simmons. This project set out to provide a framework within which teachers could examine and shape their own classroom practice according to their goals and teaching styles. This framework is built around six key ideas: *understanding performances* (or performances of understanding), *learning for understanding, teaching for understanding, understanding goals, generative topics,* and *ongoing assessment.*

Understanding performances are performances which go beyond one's rote and routine knowledge, stretching in new directions, which highlight active connection-making, and which build more understanding (Annual Report 1991–1992). Gardner (1993, p.24) explains the idea as follows:

> Understanding performances are things that kids can do every day – things that will ultimately show that they understand something about civil rights in the last hundred years – things like giving a summary of pivotal events or entering into debates or analysing newspaper articles. One of my favorite activities is taking the first page of a newspaper and saying. "Explain this article on the front page in terms of something that happened a hundred years ago in the United States." Someone who knows about American history can do that.

Gardner defines understanding as 'having a sufficient grasp of concepts, principles, or skills so that you can bring them to bear on new problems and situations,' (ibid., p.21) and this idea is captured in his newspaper example above.

Perkins and Blythe (1994, p.6) discuss their performance perspective on understanding:

> understanding is a matter of being able to do a variety of thought-demanding things with a topic – like explaining, finding evidence and examples, generalizing, applying, analogizing, and representing the topic in a new way.

Perkins and Blythe argue that some student performances, such as textbook exercises, are too routine to be considered as understanding performances because they do not take students beyond what they already know.

Perrone (1994, p.13) sees understanding in terms of empowerment:

> Our students need to be able to use knowledge, not just know about things. Understanding is about making connections among and between things, about deep and not surface knowledge, and about greater complexity, not simplicity.

Thus understanding is defined as a *demonstrable ability*. Expressed in Lampert's terms, it concerns how individual understanding is expressed to others. This definition operationalizes understanding (i.e. it can be discussed, seen and measured).

Learning for understanding occurs primarily through reflective engagement in approachable but challenging understanding performances that typically build into a chain of performances of increasing challenge and variety. Perkins and Blythe (1994, p.6) explain:

> If understanding a topic means building up performances of understanding around that topic, then the mainstay of learning for understanding must be actual engagement in those performances. The learners must spend the larger part of their time with activities that ask them to generalize, find new examples, carry out applications, and work through other understanding performances. And they must do these things in a thoughtful way, with appropriate feedback to help them to perform better.

Teaching for understanding involves organizing instruction to centre on students' reflective engagement in these activities in order to build a sufficiently full understanding over time. Teachers need to model, explain and coach students through these performances.

Understanding goals articulate the objectives for student learning (they are of the form 'Learners will understand...' or 'Learners will appreciate...'). Teaching for understanding can be planned through the thoughtful selection of understanding goals and specific understanding performances through which students develop and demonstrate their progress toward achieving the understanding goals. Teaching for understanding can also be conducted through sensitive opportunism (recognition of and capitalization on understanding goals as they emerge in classroom situations).

Generative topics are those topics and themes that provide enough depth for students to develop real understandings. A classroom might devote several weeks or even months to a generative topic, which should possess the following features: centrality to the discipline, accessibility to students, and connectability to diverse topics inside and outside the discipline. To create focus within a topic, only a few understanding goals should be defined.

A series of articles by *Project Zero* teachers and researchers explains how existing topics can be recast to make them more generative by introducing a theme or posing an evocative question. For example, 'In what sense was the Industrial Revolution progress?' enables students to consider the impact of the

Industrial Revolution on their own lives, as well as to become familiar with the events of that time and how the Industrial Revolution impacted on the lives of people then (Unger, 1994). Perrone (1994) produces topic maps to assess generative potential. He focuses on some – but not all – of the ideas in each map. To decide which mapped topic to pursue, he asks a series of questions: Which of the topics is most likely to engage my students? Is the topic central to the field of enquiry? Is it accessible as well as complex? Does it have generative potential (to the degree that it invites questions that students have about the world around them and taps the issues that students confront)?

Ongoing assessment enables students' understanding performances to be evaluated as they develop over time. The assessment approach is explained by Simmons (1994, p.22–3):

> We cannot assume that because we have taught a lesson "well," students have understood it. Rather, we have to seek evidence of understanding through student performance. ...
>
> Making the standards for good work clear to all is implicit in a classroom where students and teachers take time to reflect on and assess understanding performances. Students should know from the start the standards they are working toward. Equally important, various benchmarks of good performance can help students gauge their own understanding.
>
> In addition, students cannot achieve deep understanding if they receive evaluation passively. Taking time and energy to reflect on and improve one's work are essential to the understanding process itself.

Ongoing assessment is thus designed to serve students' learning needs (this purpose is labelled *formative* in Scottish curricular documents). The key features are the use of shared and public criteria, feedback, and frequent reflection throughout the learning process.

The framework is designed to help teachers answer the following questions about curriculum planning and classroom practice (from the Annual Report 1991–1992, p.44):

> What ideas, concepts, themes, topics, events do we want students to understand? And what aspects of those concepts do we want students to focus on? Of all the various kinds of things students might do to build and demonstrate their understandings of the chosen topics, which performances would be the most beneficial to them? And how do we measure progress toward achieving understanding?

Four key ideas within the framework – generative topics, understanding goals, understanding performances and ongoing assessment – enable teachers to address these questions. Perkins and Blythe (1994) discuss ways in which the framework can help teachers make their efforts to teach for understanding more successful. They identify that in some cases there is an insufficient presence of understanding performances, in others the curriculum lacks the focus provided by thinking in terms of carefully selected generative topics and goals for understanding, and in others students do not receive the ongoing assessment needed to help them learn from the performance of understanding.

These four questions are pretty fundamental. Is such a radical re-appraisal of school education justified? Perkins and Blythe (1994, p.5) argue that it is, and that teachers and school administrators need to weigh up carefully the importance of teaching for understanding:

> We firmly believe that understanding deserves special attention. This does not mean that we deny the importance of other educational goals. For instance, a number of routine skills regarding arithmetic, spelling and grammar certainly need development. But what use are students to make of the history or mathematics they have learned unless they have understood it?

Gardner (1993, p.23) talks of three kinds of understanding. The first kind is intuitive ('The young child has wonderful theories about the world and applies them promiscuously...'). The second kind is possessed by the scholastic learner, a person whose knowledge can be used only in a very explicit context ('If you ask a certain question – using the right words – you get the right answer. But if you meet the student on the street after the exam is over and ask a slightly different question, the answer you'll get will be wrong.'). And the third kind?

> Finally, there's our hero: the person who has knowledge and knows when to use it and when not to use it. That expertise constitutes understanding.

Gardner cites evidence from a range of research studies in which the 'best' students from prestigious universities could not apply the knowledge they had acquired in school to answer questions (such as identifying the forces acting on a coin when it reaches the mid-way point on its upward trajectory). The students exhibited the same misconceptions and misunderstandings primary school children have evolved to make sense of the world around them.

Gardner views the failure of American schools to educate for understanding as being unwitting rather than deliberate. He identifies a range of obstacles to understanding:

- The 'correct-answer compromise'. Students learn to give certain answers in certain situations – answering a question on a multiple-choice test or carrying out a problem in a specified way – and everybody is happy:

  A certain kind of performance is accepted as adequate, and the gap between what passes as understanding and genuine understanding remains great. (ibid., p.22)

- 'Pressure for coverage':

  The greatest enemy of understanding is coverage – I can't repeat that often enough. If you're determined to cover lots and lots of things, you are guaranteed that most kids will not understand, because they haven't had time enough to go into things in depth, to figure out what the requisite understanding is, and to be able to perform that understanding in different situations. (ibid., p.24)

- 'Short-answer assessments' (a partner to coverage) in which students are asked to give brief answers on aspects they have already been coached on: 'The problem is that life isn't a multiple-choice test, and it doesn't come with single answers.' (ibid., p.22)

- There is a failure to correct children's misconceptions and immature understandings, for example, in relation to the 'scientific' theories that they form. These theories are powerful and difficult to eradicate, and some aspects of school learning are very counterintuitive and therefore difficult to learn.

- Institutional constraints (such as class size).

- Disciplinary constraints (see also Gardner and Boix-Mansilla, 1994). A good teacher has to help students appreciate that what counts as cause and effect, data and explanation, use of language and argument, varies across the disciplines.

Gardner concludes that the implications of 'teaching for understanding' are far-reaching. It isn't just a matter of introducing performance-based assessments:

> You can have terrific assessments, but if you don't have a curriculum that nurtures understanding, if you don't have teachers who help kids look at things in different kinds of ways, the assessments are worthless. (ibid., p.24)

A particular concern raised by the *Project Zero* team is that teaching loads in American schools are too full and leave very little time for curriculum planning, development of appropriate and challenging materials, and critical reflection or collective thought. (Annual Report 1991–2, p.3)

2.3.3 *Discussion on teaching for understanding*

There are six related points that I wish to raise stemming from comparisons between the methods of 'teaching for understanding' discussed in this section, and the infusion methods discussed in the previous section, before reaching a general conclusion.

When the *Project Zero* investigators discuss methods of teaching for understanding, they do not place an explicit emphasis on the thinking skills and processes that play such an important role in understanding performances. Swartz and Park's advice that teachers must take time to clarify the skilful thinking needed in order to yield more thoughtful responses from students, must surely be heeded if the maximum benefit is to be derived for student learning. This advice may be heeded in practice through the methods of on-going assessment, or the choice of a generative topic that focuses on an important 'way of knowing' within a discipline (such as evaluating sources in history), or the specification of an understanding goal that focuses explicitly on some aspect of thinking or problem solving. But the systematic focus on thinking skills or processes that is evident in the two examples of infusion methods may be lost. Thinking is a much more highly differentiated concept than understanding (see, for example, Ennis's (1987) taxonomy of critical thinking dispositions and abilities) and is also more active – it involves problem solving, decision making, and so on. Therefore an emphasis on understanding should not be allowed to supplant an emphasis on thinking skills.

A second point concerns the operational definition of understanding. Schoenfeld (1985, p.14) warns that holding a view of problem solving as an operational definition of understanding is too narrow. He argues that the issues are more complex in mathematical problem solving:

> One must deal with (1) whatever mathematical information problem solvers understand or misunderstand, and might bring to bear on a problem; (2) techniques they have (or lack) for making progress when things look bleak; (3) the way they use, or fail to use, the information at their disposal; and (4) their mathematical world view, which determines the ways that the knowledge in the first three categories is used.

Thus a student's failure to obtain a solution to a problem cannot be straightforwardly attributed to his or her failure to understand the requisite information; other factors may come into play. This highlights the critical importance of ongoing (formative) assessment – an element in the *Project Zero* framework and the aspect which Lampert chooses to focus upon when she expands her definition of understanding as 'sense making'. Without this element, one is left to guess at the causes of students' difficulties on the evidence of summative measures alone, and the opportunity for students to redirect their efforts or thinking during the course of the work has been lost.

A third point concerns the selection of appropriate lesson or problem contexts. Three of the four questions that the *Project Zero* framework is designed to address focus on the choice of contexts for developing students' understanding (what ideas, concepts etc. do we want students to understand? what aspects of these? which performances would be most beneficial?). Lampert discusses the need to ask questions which will challenge children's assumptions in order to develop their understanding. Polya and Schoenfeld stress the nature of the problems that students should encounter, and Swartz and Parks devote a chapter in their teachers' handbook to selecting contexts for infusion lessons. The need for the curriculum to have a more selective focus, rather than to attempt to be all encompassing, lies behind these concerns. It is the main solution to the problem of the content-packed curriculum.

Within *Project Zero*, the lack of connectedness in the school curriculum is the issue that has most prompted its re-appraisal:

> My math teacher in high school – a very good teacher – spent significant time teaching me and the rest of the class about quadratic equations. Almost everyone I know today learned how to handle quadratic equations at some point. Yet most of these folks seem to have had little use for them lately. Most have probably forgotten what they once knew about them.
>
> The problem is, for students not headed in certain technical directions, quadratic equations are a poor investment in understanding. And the problem is much larger than quadratic equations. A good deal of the typical curriculum does not connect – not to practical applications, nor to personal insights, nor to much of anything else. It's not the kind of knowledge that would connect. Or it's not taught in a way that would help learners to make connections. (Perkins, 1993, p.32)

If, however, the generative potential of a topic is assessed too narrowly (e.g. in terms of the things that students are interested in), or if it proves difficult for student projects to demonstrate practical applications, there is a danger that, in

a chain of connections, a vital early link may be removed. Who is to say in which direction students are headed, and how early this is to be decided? Who can tell (in the spirit of Polya) which 'discovery' will spark a life-long interest in a topic? Transfer is too unpredictable a phenomenon to lay bets on this.

A fourth point concerns the depth and duration of study on a generative topic, which, it is suggested, could extend to about two months. Students are not all equally interested in the same topics and if a topic continues for too long this could kill some students' interest in the themes that the topic explores. There is clearly a need to monitor students' affective responses, to cater (to some extent) for individual student choice, and to find an appropriate balance between depth and breadth of study. (It emerges in the next section that varied practice, which demands breadth in the curriculum, is helpful to foster transfer.)

A fifth point concerns how teachers who are locked into existing syllabuses can make time to explore important topics in depth. An anticipated benefit of students being able to perceive the connections between subject matters is that instruction should go more quickly in related domains (Perkins and Salomon, 1989). This enables teachers to spend longer on the central aspects of their subject. Perrone (1994, p.13) discusses how teachers can make time through prioritizing:

> In most places, we have found that district guidelines haven't kept teachers from doing what they felt was important. In an extreme case, several teachers in a southern state with heavy state mandates and tests did understanding-oriented teaching work Monday through Thursday and devoted Friday to what they called "Caesar's work."

There is an element of risk taking involved for teachers when going down this route for the first time – what if students fail to see the connections between subject matters or fail to learn the material that isn't prioritized (perhaps because they find it uninteresting or it is presented too hurriedly)?

A sixth and final point, which I shall state briefly, is that there can be no real progress towards infusing thinking skills across the curriculum or teaching for understanding unless teachers are given the time and opportunity to engage fully with the ideas, and support to implement them.

In spite of these reservations, it is clear that the explicit focus on teaching for understanding is a necessary counter to an over-salient curriculum on the one hand, and the possibility of a shallow adoption of 'infusion' methods, resulting in superficial treatment of subject content, on the other.

2.4 TEACHING FOR TRANSFER

A theme that runs throughout the discussion in the two previous sections is transfer of learning to new contexts. This section examines the question of whether teaching needs to be directed specifically to fostering transfer, the theoretical arguments on transfer, possible transfer mechanisms and methods of teaching for transfer.

2.4.1 Does teaching need to be directed to fostering transfer?

From Resnick's conclusion (see 2.2.3), in which she argues that no separate brief need be made for teaching higher order skills when these are successfully embedded in academic disciplines, it is not apparent that there is any special need to teach skills in a manner that will promote their transfer to new contexts.

However Perkins (1993, p.32) argues that if we want students to be putting to work in diverse settings the understandings they acquire, then we need to teach explicitly for transfer by, 'helping students to make the connections they otherwise might not make, and helping them to cultivate mental habits of connection-making.' He views the two agendas of teaching for transfer and teaching for understanding as closely allied:

> an understanding performance virtually by definition requires a modicum of transfer, because it asks the learner to go beyond the information given, tackling some task of justification, explanation, example-finding or the like that reaches further than anything in the textbook or lecture. (ibid., p.32)

Also many understanding performances transcend the boundaries of the topic, the discipline or the classroom. Therefore to teach for a full and rich understanding requires reaching well beyond the obvious and conventional boundaries of the topic.

2.4.2 Theoretical arguments

An important underlying theoretical issue is the relationship between what Perkins and Salomon (1989) term specialized domain knowledge and general strategic knowledge. Resnick (1987) discusses the issue in terms of knowledge that is specific to a particular domain – in the form of both specific facts and organizing principles, and 'enabling skills' for learning and thinking – aspects including a wide range of oral and written communication skills, mathematization and representational abilities, principles of reasoning, and skills of argument construction and evaluation.

In an article entitled, 'Are cognitive skills context-bound?', Perkins and Salomon (1989, p.16) introduce the issue as follows:

> Effective problem solving, sound decision making, insightful invention – do such aspects of good thinking depend more on deep expertise in a specialty than on reflective awareness and general strategies?

They suggest that the interaction between general strategic knowledge and specialized domain knowledge has been over-simplified by those who adopt either a strong specialist or strong generalist position.

Those who adopt a strong specialist position would argue that skilful, thought-demanding performance is relatively context-bound and depends upon the possession of a rich knowledge base in the domain. They would therefore propose a curriculum in which a richly developed local knowledge base is developed, subject by subject. Those who adopt a strong generalist position would argue that skilful, thought-demanding performance reflects the use of general abilities of some sort. They would propose a curriculum in which 'transferable' skills, such as problem solving and self-management, are developed.

By tracing developments over thirty years, Perkins and Salomon build a case for viewing general strategic knowledge and specialized domain knowledge as 'close partners'. They begin by summarizing the arguments in favour of a generalist position. They then present three counter-arguments deriving from studies on (a) expert vs. novice performance; (b) general heuristics, applied to Artificial Intelligence (AI) and education; and (c) transfer. Finally they present a synthesis position which draws on the most recent research findings.

Other reviewers (e.g. Frederiksen, 1984; Resnick, 1987; Nisbet and Davies, 1990; De Corte, 1990; Bransford and Vye, 1989) have also examined this issue through tracing the changing influences of ideas from various fields – cognitive psychology, philosophy, information technology and education – on how the teaching of thinking is viewed. Below I present, in a highly summarized form, the main arguments, drawing mainly on Perkins and Salomon's analysis.

2.4.2.1 Summarizing the arguments in favour of a generalist position

It was widely thought, before the 1950's, that good problem solving and other intellectual performances reflected general strategies operating on whatever database of knowledge happened to be needed. Many of Polya's heuristics were plainly applicable to problems of all sorts, which encouraged the view of

problem solving as a general ability and mathematical problem solving as a special case. An experiment in AI produced the 'General Problem Solver' (GPS) – developed by Newell, Shaw and Simon around 1957 – a program which relied on a flexible means-ends analysis. Input to the program included information about a beginning state, an end state and allowable operations. The program pursued a chain of operations for transforming the beginning state into the end state, each chosen to reduce the contrast between the current state and the end state. Here again, it appeared that problem solving power lay in some rather general principles, systematically applied to whatever the relevant database of knowledge happened to be. Local knowledge was thought to be not very important; there wasn't much to it beyond a few rules or axioms.

2.4.2.2 *Three counter-arguments in favour of a specialist position*

This picture began to break down as more and more studies presented contrary findings. The first counter argument came from studies on expertise. Investigators were discovering that the seeming smallness of the database demanded by chess, symbolic logic, and other favourite areas of research was deceptive:

> Research on the games of grand master chess players showed that their tactics depended on an enormous knowledge base of important patterns of chess pieces. ... Expert chess players reasoned about the game using these chunk-like configurations, rather than thinking about one piece at a time, and so had much more power to think ahead and devise strategies than a simple command of the rules would afford.
>
> (Perkins and Salomon, 1989, p.18)

From a wide range of studies (e.g. Larkin (1980) on problem solving in physics, and Soloway and Ehrlich (1984) on computer programming) a general profile of expertise began to emerge. Expert performance entailed: a large knowledge base of domain-specific patterns (or schemata); rapid recognition of situations where these patterns apply; and reasoning that moves from such recognition directly toward a solution (forward reasoning). In contrast, novices tended not to see the relevant patterns, they often based their reasoning on superficial problem content, and reasoned backwards from the unknown.

> General heuristics appeared to be no substitute for the rich database of ramifications, stored in memory, accessed by recognition processes, and ready to go. Indeed, the broad heuristic structure of expert as contrasted to novice problem solving – the reasoning forward rather than reasoning backward – seemed attributable not to any heuristic sophistication on the part of the experts, but to the driving influence of the experts' rich database. (Perkins and Salomon, 1989, p.18)

The second counter argument came from studies on general heuristics, applied to teaching and AI. Schoenfeld (1985, p.71–2) comments of the history of applying Polya's heuristics to mathematical problem solving:

> Once the most important heuristic strategies have been discovered and elaborated, the next step is obvious. One should provide direct instruction in these strategies, thereby saving students the trouble of having to discover the strategies on their own. ... There is but one problem with this rationale: It hasn't worked, and not for lack of trying. The literature of mathematics education is chock-full of heuristic studies. Most of them, while encouraging, have provided little concrete evidence that heuristics have the power that the experimenters hoped they would have. ... In brief, heuristics have proven far more complex and far less tractable than had been hoped or expected.

Although students understood the heuristics in broad terms, they lacked a firm foundation in mathematics to be able to apply them.

Similarly, in AI applications, generic programs (like GPS) performed poorly in complex problem solving domains such as medical diagnosis, whereas programs designed specifically for those domains (expert systems) were very successful. Investigators in the AI community came to refer to general heuristics as *weak methods*[8] and the phrase *power-generality trade-off* was coined.

The third counter argument came from studies on transfer. Studies ranging across many domains failed to uphold predictions on transfer. Thus, on transfer from programming: 'Investigations of the impact of programming instruction on cognitive skills have yielded occasional positive and many negative findings.' (Salomon and Perkins, 1987, p.149) Nisbet and Davies (1990) suggest that the explanation may lie in the difficulty that students experience with programming and the failure of those who teach it to extract transferable principles in their concern to teach limited techniques.

In general, students do not carry over the facts and principles they acquire in one context into another context. Knowledge tends to get glued to the narrow circumstances of initial acquisition (Perkins, 1993). This may be because knowledge is more 'local' in character than one would imagine, although other explanations are possible (Perkins and Salomon, 1988).

---

8 Newell (1980, p.186) lists the most well known of these, including 'generate and test', 'divide and conquer' and means-ends analysis.

Paul (1987, p.134) refers to *inert knowledge*, 'knowledge that we in some sense *have* but do not use when logically relevant, knowledge that just sits there in our minds, as it were, without activating force.' Inert knowledge is typically viewed as a problem of transfer.

2.4.2.3 *Towards a synthesis position*

From recent (1980's onwards) studies on expertise it emerges that a number of general heuristics not apparent when experts face typical problems play a prominent role when experts face atypical (i.e. within the domain but not 'textbookish') problems. Perkins and Salomon (1989, p.20) use the results of a study by Clement on physics problem solving to illustrate this:

> As in other work on expertise, the experts addressing such problems certainly use their rich physics knowledge base, trying to see the deep structure of the problem and deploying principles like conservation of energy. But, because these unusual problems do not yield to the most straightforward approaches, the experts also apply many general strategies. For example, the experts faced with an unfamiliar problem will often: (a) resort to analogies with systems they understand better; (b) search for potential misanalogies in the analogy; (c) refer to intuitive mental models based on visual and kinesthetic intuition to try to understand how the target system would behave; (d) investigate the target system with "extreme case" arguments, probing how it would work if various parameters were pushed to zero or infinity; (e) construct a simpler problem of the same sort, in the hope of solving that and importing the solution to the original problem.

When extended to how people learn in new domains, Resnick (1987, p.45–6) summarizes the evidence as follows:

> Generally speaking, people rely on powerful but only narrowly applicable thinking methods in domains in which they are expert and use broadly applicable but weak methods for learning and thinking in fields they know little about. Good thinkers need both the powerful but specific and the general but weak kinds of skills.

There is also evidence from various studies of students' successful use of general strategies following teaching interventions. Schoenfeld's work on teaching heuristic strategies for mathematical problem solving provides one such example (see 2.2.5.2). A second example is Brown and Palincsar's reciprocal teaching method for developing self-regulated reading, which encourages and refines four key strategies - questioning, clarifying, summarizing and predicting. As a result of this intervention, students' reading comprehension improved markedly, the improvements were maintained over time, and transfer was demonstrated to in-class reading in science and

social studies (Brown and Palincsar, 1989; Palincsar and Brown, 1989). A third example is Scardamalia and Bereiter's method for developing writing skills, which supplants the novices' 'knowledge-telling' strategy (when given a topic to write on, they immediately produce text by writing their first idea, then their next, and so on, until they run out of ideas) with a strategy which incorporates the linear generation of text but is organized around a more complex structure of goal setting and problem solving. This strategy splits the planning process into five general goals, thus reducing the cognitive load on the learner: (a) generating a new idea; (b) improving an idea; (c) elaborating an idea; (d) identifying goals; and (e) putting ideas into a cohesive whole. Specific prompts have been developed for each goal, for example, on elaborating an idea, 'An example of this is...', 'This is true, but it's not sufficient so...', 'My own feelings about this are...', and so on (Collins, Brown and Newman, 1989, p.464–9).

Fresh appraisals of the often mixed findings from studies on transfer across a range of domains led to the identification of *conditions for transfer*. For example, Schoenfeld (1985) discusses breaking down general heuristic strategies into a collection of coherent substrategies which are then carefully delineated, discussing the conditions under which they seem to be appropriate, and using problems from new domains for practice, with later discussion as to how and why the particular strategies used were appropriate.

2.4.2.4 *A synthesis position*

Perkins and Salomon (1989), on the basis of this evidence and their general observation that, 'Some people seem generally smart – not just knowledgeable, but insightful no matter the subject,' (p.19) propose a synthesis position:

> In the synthesis, general cognitive skills do not function by somehow taking the place of domain-specific knowledge, nor by operating exactly the same way from domain to domain. Rather, cognitive skills are general tools in much the way the human hand is. Your hands alone are not enough; you need objects to grasp. ... Likewise, general cognitive skills can be thought of as general gripping devices for retrieving and wielding domain-specific knowledge, as hands that need pieces of knowledge to grip and wield and that need to configure to the kinds of knowledge in question. (ibid., p.23)

Thus, taking an example:

> As you learn a new subject matter, trying to think of counterexamples to claims surely is a good critical posture to maintain. But you have to accumulate knowledge in the domain with which to find or build counterexamples. And you have to develop a sense of what *counts* as a counterexample in the domain. (ibid., p.23)

In the synthesis general cognitive skills always function in highly contextualized ways. General, learnable and worthwhile cognitive skills can be recognized by their seeming use (e.g. a philosopher's bent to seek counterexamples); important role (e.g. seeking counterexamples exposes flaws in claims); transferability, and common absence (everyday experience tells us that most people don't do it).

They note that some educators (e.g. Hirsch) have taken the negative arguments from expertise, weak methods and transfer to eschew attention to higher order skills so that more time can be given to building factual knowledge. Unfortunately, this position forces a choice between 'cultural literacy' and critical thinking (Perkins and Salomon, 1988; Otto, 1990). They comment:

> To be sure, general heuristics that fail to make contact with a rich domain specific knowledge base are *weak*. But when a domain-specific knowledge base operates without general heuristics, it is *brittle* – it serves mostly in handling formulaic problems. (Perkins and Salomon, 1989, p.23)

Perkins (in Brandt, 1990) asserts that any subject matter has a particular content, involves some strategies and skills specific to it, and invites application of some 'thinking organizers' (patterns of thinking) that cut across different subject matters – looking for the other side of the case, breaking a problem into parts, comparing and contrasting, and so on. General thinking organizers help the learner to apply specific knowledge flexibly and generatively.

The synthesis position leads logically towards infusion of thinking skills into the curriculum (but does not exclude the possibility of a separate 'thinking skills' insert to support infusion) and 'teaching for transfer', to enable general strategies to configure to the knowledge base in the new domain.

2.4.3 *Transfer mechanisms*

The following ways of discussing transfer mechanisms are commonly found in the literature.

2.4.3.1 *Robustness*

Transfer can prove to be more or less robust. It may occur spontaneously, or if the learner is alerted to the relevance of a general strategy in the new domain, or (at its least robust) if instruction systematically helps the learner to contextualize the method in the new domain (Perkins and Salomon, 1989).

2.4.3.2 *Potentials of transfer*

The discussion above focuses mainly on transfer of general strategies, however other aspects of learning can also transfer. The transfer of basic skills (literacy, numeracy etc.) is a routine target of schooling, the knowledge that students gain in school ought to inform their thinking across the board, and the transfer of thinking skills should enable students to become better creative and critical thinkers in the many contexts that call for a thoughtful approach. Also there are different levels of generality in any subject matter. Many patterns of thinking are of intermediate generality, cut across certain domains, and provide natural prospects for transfer, but conventional subject matter boundaries may inhibit the emergence of these patterns (Perkins and Salomon, 1988).

2.4.3.3 *Positive and negative transfer*

Transfer may enhance performance in a new domain (positive transfer) or interfere with it (negative transfer):

> people commonly ignore the novelty in a situation, assimilating it into well-rehearsed schemata and mindlessly bringing to bear inappropriate knowledge and skill, yielding negative transfer.
>
> (Perkins and Salomon, 1989, p.22)

2.4.3.4 *Near and far transfer*

Transfer may be 'near' or 'far', depending on the size of the step. A short step would be from driving a car to driving a truck, a long step would be from chess playing to planning a political campaign.

A short step does not guarantee that transfer will occur, for example, in the context of programming: 'a considerable portion of beginning students' knowledge of commands in a programming language is inert even in the context of active programming, where there is hardly any gap to transfer across.' (Perkins and Salomon, 1988, p.23–4)

2.4.3.5 *Transfer may operate by either of two mechanisms*

The two examples above (driving a car to driving a truck, chess playing to planning a political campaign) suggest two very different transfer mechanisms:

> Low road transfer reflects the automatic triggering of well-practiced routines in circumstances where there is considerable perceptual similarity to the original learning context. ...

> High road transfer depends on deliberate mindful abstraction of skill or knowledge from one context for application in another.
> (Perkins and Salomon, 1988, p.25, 27)

Identification of these mechanisms clarifies the conditions under which different sorts of transfer occur. The conditions for low road transfer of some basic skills – reading, writing and arithmetic – are generally met in schools. Such skills are practised in a variety of contexts to (near) automaticity. Low road transfer is not confined to basic skills. Aspects such as defining a problem carefully before embarking on a solution can also transfer by the low road if the conditions for such transfer are met.

The conditions for high road transfer are frequently not met in schools. There is the question of initial learning. One cannot expect transfer of a performance that has not been learned, for example:

> the skill students have learned through their study of history is not the skill they need when they consider today's newspapers. We want them to make thoughtful interpretations of current events, but they have learned to remember and retrieve knowledge on cue.
> (Perkins and Salomon, 1988, p.28)

Conventional instruction often emphasizes the particulars of situations (e.g. the story of particular historical episodes). It does little to decontextualize the patterns or general principles, in order to break them free of their accidental associations in particular settings. Transfer by the high road is effortful, and therefore motivation is a factor, and it requires some mental skill.

2.4.3.6 *High road transfer may be forward reaching or backward reaching*

The processes of abstraction and connection-making may involve reaching forwards – learning something and abstracting it in preparation for applications elsewhere – or reaching backwards – when facing a particular problem, abstracting the key characteristics of the situation and looking backward into one's experience for matches, for example, by making fertile analogies (Perkins and Salomon, 1988).

2.4.4 *Methods of 'teaching for transfer'*

2.4.4.1 *'Hugging' and 'bridging'*

Perkins and Salomon (1988, p.28–9) propose two techniques for promoting transfer, 'hugging' and 'bridging':

> 'Hugging' means teaching so as to better meet the resemblance conditions for low road transfer. Teachers who would like students to use their knowledge of biology in thinking about current ecological problems might introduce that knowledge in the first place in the context of such problems. ...
>
> 'Bridging' means teaching so as to meet better the conditions for high road transfer. Rather than expecting students to achieve transfer spontaneously, one "mediates" the needed processes of abstraction and connection-making. ... For example, teachers can point out explicitly the more general principles behind particular skills or knowledge, or, better, provoke students to attempt such generalizations themselves...

Although most teachers do use occasional hugging and bridging techniques, rarely is it done systematically enough to make an impact. The overwhelming emphasis on subject-matter specific aspects swamps the effect.

Even without bridging, hugging can make a substantial impact on transfer. For example, inert knowledge has been a serious problem in medical education where traditionally students memorize multitudinous details of anatomy and physiology outside the context of real diagnostic application. Problem-based learning, in which students gain their technical knowledge in the context of working through case studies demanding diagnosis, helps to overcome this difficulty (Perkins and Salomon, 1988). An article entitled 'Tomorrow's Doctors: Learning Skills and Attitudes' (University of Glasgow Report, 1996–7, p.12) begins with a justification for problem-based learning: 'What is it that changes an 18-year-old school leaver into a 23-year-old doctor? ... Skills and attitudes. It is the ability to reason and apply knowledge, to talk and listen to patients, to examine patients, to reassure them, to get information from them and to use the information gained and the underpinning knowledge together in a synthesis that bears on their diagnosis and treatment. That is far more important than learning endless facts.'

2.4.4.2 *Metacognition makes a particular contribution to transfer*

The potential for self-regulating strategies to transfer was highlighted by Schoenfeld (see 2.2.5.2). The 'metacognitive map' which students create in the *thinking about thinking* phase of infusion lessons (Swartz and Parks, 1994) is the means by which students distance themselves from the lesson's content so as to consider the thinking they just did, in preparation for applying it more widely.

Nisbet and Shucksmith (1984) highlight the particular contribution which metacognition can provide to improving the capacity to transfer. Awareness of

one's own mental processes, and skill and practice in self-monitoring are the means by which students can decontextualize the instruction, enabling the learner to take a more active and responsible role in the learning process. Perkins and Salomon (1988) discuss the issue in terms of teaching students how to *learn for transfer*, so that they do the hugging and bridging for themselves. This important theme will be developed more fully in the next section.

2.4.4.3 *Attitudes and beliefs come into the equation*

Paul (1987, p.134) argues:

> Children do not *transfer* the knowledge they learn in school to new settings because they already have activated ideas and beliefs in place to use in those settings. ... Only by bringing out the child's own ideas in dialogical/dialectical settings can the child begin to reconstruct and progressively transcend these conceptions.

Lampert's choice of questions to challenge pupils' assumptions (see 2.3.1) and her methods of questioning are also in this vein.

Schoenfeld (1985) discusses the impact of students' belief systems on their behaviours. If students believe that formal mathematics has little or nothing to do with real thinking or problem solving, then if a problem calls for discovery, formal mathematics will not be invoked. If students believe that mathematics problems can always be solved within ten minutes, then they will not persist. If students believe that only geniuses are capable of discovering or creating mathematics, then if they forget something, too bad, they cannot derive it, and if they are given a procedure to use, they will accept it at face value without trying to understand how it works. He recommends problem-based learning to enable students to develop a mathematical viewpoint (see 2.2.5.2).

2.5 'LEARNING TO LEARN'

2.5.1 *Introduction*

In this section the focus shifts to learning skills and strategies and the role of metacognition in learning to learn. The discussion is strongly influenced by Nisbet and Shucksmith's (1986) view of how 'learning to learn' must be done:

> Learning to learn must be done in a real context: the best way to acquire strategies of learning is in the process of learning. But if what is learned is to be transferable to other contexts, then it must be taught in such a way as to encourage transfer. (p.94)

They provide a simple definition of learning strategies as, 'the processes that underlie performance on thinking tasks,' and use an analogy (concerning football tactics) to explore the concept further (ibid., p.24).

Sternberg (1987) refers to *executive processes* which are used to plan, monitor and evaluate problem solving performance. He has developed a theory of human intelligence in which executive processes play an important role. The theory forms the basis of a programme to train intellectual skills. He describes part of the theory as follows:

> The first, componential part of the triarchic theory specifies three basic kinds of information-processing components: (1) metacomponents, which are executive processes used to plan, monitor and evaluate one's strategy for solving problems; (2) performance components, which are nonexecutive processes used to execute the instructions of the metacomponents for solving problems; and (3) knowledge-acquisition components, which are nonexecutive processes used to learn how to solve the problems in the first place. (p.198)

Unfortunately, the language in which the ideas are expressed makes the theory very difficult to comprehend. Nisbet and Shucksmith (1984) complain that the information-processing terminology used by cognitive psychologists is complex and forbidding, and to this I would add, confusing – how can a nonexecutive process execute something? They comment: 'Not surprisingly, this kind of research has had relatively little impact on what actually happens in schools.' (p.2) On the other hand, they argue, developments in schools in the form of teaching study skills operate on simplistic assumptions, having their basis in 'good practice' and lacking any sound rationale. Biggs (1988) comments that the typical outcome of such courses, which teach procedures such as note-taking, recitation, self-testing, scheduling and the like, is initial enthusiasm, a possible temporary improvement in achievement, and then reversion to original habits.

2.5.2 *Teaching skills and strategies*

Nisbet and Shucksmith (1984) distinguish between skills and strategies by defining strategies as being a level above skills. Many skills are taught and learned in the context of specific subjects or situations, and some general skills are specific to tasks shared in related subjects. Examples of such skills are using a lathe, dictionary or computer. A strategy involves putting skills together with a conscious purpose, not just on instruction or demand. A strategy is more modifiable and flexible in nature than a skill, which is more 'reflexive'. In order to apply a strategy you must possess a range of skills, be aware of a range

of possible strategies, and select appropriately from these. This implies the capacity to transfer skills and strategies to fit new situations.

They suggest that the skills should be taught in such a way as to cultivate the more general strategies, and similarly the strategies should be integrated to cultivate an *approach to learning* which involves self-regulation of learning, self-monitoring selection of strategies and insight through reflection. In this way, learning to learn is, 'embedded firmly in mainstream teaching, and not imparted as an extra in one or two inspiring talks or vague exhortations as part of a study skills curriculum.' (ibid., p.9)

They identify a set of learning strategies commonly mentioned in the literature: orientation, planning, monitoring, checking, revising and self-testing. Although many teachers encourage these strategies, they tend to be taught in specific contexts – planning for projects, checking for arithmetic, self-testing for spelling, and so on – and children are rarely taught to generalize them. Of four criteria that they identify for skills training – mastery, retention, durability and transfer – only the first three seem to receive much attention. However when Nisbet and Shucksmith set out to find examples of classrooms where children were being set significant challenges which would develop their use of strategic thinking, they were disappointed:

> there were many time-filling tasks and much "busy work" which did not demand mental effort. There was reinforcement and over-learning, but we did not often find the teaching or encouragement of strategies which we were looking for. Of the criteria ... we found emphasis on mastery and retention, and sometimes durability, but not transfer. (ibid., p.12–3)

For learning strategies to transfer, they must first be consciously articulated and then practised so that eventually they become part of an habitual unconscious approach to learning. Conscious articulation involves identifying the variety of sub-skills which are then built into sequences or strategies for tackling problems. Practice is necessary, since otherwise by constantly having to think about what you are doing you may inhibit your capacity to do it.

The methods that Nisbet and Shucksmith propose for improving children's capacity to think and work strategically comprise direct teaching of strategies, modelling of strategies and encouraging metacognitive skills. The key point about direct teaching is that any procedures must not be taught as tricks to cope with specific tasks: teaching must stress the potential for transfer rather than being mechanistic. Reconciling the requirement for flexibility with the need for practice so that procedures become automated is, however, no easy task.

Modelling involves the teacher sharing with students how his or her own learning, the task and the learning context influences performance:

> The purpose is not to "beat the child over the head with a new skill", but to show how to "select skills within the context of natural predicaments". (ibid., p.14)

Students should also be allowed to explore their own metacognitive knowledge by discussion and exposure to a variety of contexts or circumstances where it is called into play. The role of metacognition in 'learning to learn' is discussed below.

The above advice forms part of a framework which links cognitive goals (tasks and problems), metacognitive knowledge, and learning strategies (Nisbet and Shucksmith, 1986, p.95). Within this framework, teachers can intervene to make learning more effective by:

- establishing clearer cognitive goals and sharing them with students; distinguishing between the outcomes and processes of learning; dividing tasks into constituent parts; relating goals to pre-planning and subsequent reflection sessions with pupils.

- developing students' metacognitive knowledge through modelling and exploration.

- attempting to discern general strategies used across tasks and distinguishing these from skills; stressing common strategic elements and reinforcing their successful use where possible; identifying how strategies change in line with goals, knowledge and context; encouraging children to generate search procedures that enable them to make full use of their available strategic repertoire.

2.5.3 The role of metacognition in learning to learn

John Flavell at Stanford University is credited with introducing the term 'metacognition' in 1970 to describe the monitoring of one's thinking. He defines and explains the concept as follows:

> "Metacognition" refers to one's knowledge concerning one's own cognitive processes ... For example, I am engaged in metacognition ... if I notice that I am having more trouble learning *A* than *B*; if it strikes me that I should double-check *C* before accepting it as a fact; if it occurs to me that I had better scrutinize each and every alternative in any multiple-choice type task situation before deciding which is the best one; if I sense that I had better make a note of *D* because I may forget it ... Metacognition

> refers ... to the active monitoring and consequent regulation and orchestration of these processes ... usually in the service of some concrete goal or objective. (Flavell, 1976, p.232)

More recently, Flavell (1979) has distinguished metacognitive knowledge (relating to persons, tasks and strategies), metacognitive experiences, goals (or tasks) and actions (or strategies) as four interacting components of cognitive monitoring. The first example in the quotation above is in the category of metacognitive experiences; each of the other examples demonstrates the activation of a strategy arising from metacognitive knowledge and/or experiences. These are somewhat fine distinctions, as the following (partial) example reveals:

> Let us begin at the point where some self-imposed or externally imposed task and goal are established. Your existing metacognitive knowledge concerning this class of goals leads to the conscious metacognitive experience that this goal will be difficult to achieve. That metacognitive experience, combined with additional metacognitive knowledge, causes you to select and use the cognitive strategy of asking questions of knowledgeable other people. Their answers to your questions trigger additional metacognitive experiences about how the endeavor is faring. These experiences, again informed and guided by pertinent metacognitive knowledge, instigate the metacognitive strategies of surveying all that you have learned to see if it fits together into a coherent whole, if it seems plausible and consistent with your prior knowledge and expectations, and if it provides an avenue to the goal. This survey turns up difficulties ... (Flavell, 1979, p.909)

Suffice it to say that this level of detailed analysis will not form part of the work of this thesis. The ability to distinguish between a cognitive and metacognitive strategy rests with identifying the reason for invoking it. If its purpose is self-monitoring or self-regulation of learning, then I shall use the label 'metacognitive'. (Brown (1984) considers the usefulness and practicality of these distinctions, for example, does, 'I feel that I'm getting nowhere with this essay,' represent an experience, or the sad but all too frequent knowledge – 'I am getting nowhere with this essay'.)

Flavell suggests that metacognitive experiences (a kind of quality control on thinking) are especially likely to occur in situations that stimulate a lot of careful, highly conscious thinking, such as when a school task expressly demands that kind of thinking, in novel situations where every major step you take requires planning beforehand and evaluation afterwards, and where high affective arousal or other inhibitors of reflective thinking are absent. The question of the effect of context and stimuli in invoking metacognition and

hence strategic activity is considered by Nisbet and Shucksmith (1986) to be of crucial importance for the classroom.

2.6 THE AFFECTIVE AND SOCIAL DIMENSIONS OF LEARNING AND THINKING

Although, in this section, three separate headings have been identified – the affective dimension of learning, motivation, and the influence of the social setting – the ideas are very inter-related.

2.6.1 The affective dimension of learning

Considerable emphasis is given, in this thesis, to pupils' affective responses to the learning environment (all of chapter 7 is devoted to this topic). Why are these important? Nisbet (1990, p.4) highlights the central role of emotions and feelings in thinking:

> We need to build on the satisfactions of thinking, not be preoccupied with the difficulties or the shame of failure. Resolving a difficulty, understanding a complex topic, the flash of insight in solving a problem: these are (or should be) deeply satisfying experiences. Confusion and inability to comprehend are frustrating and quickly erode the will to learn or to try.

Goleman's recent best-seller, 'Emotional Intelligence', has helped to raise awareness in Scottish schools of the crucial role of emotions and feelings in children's learning:

> when too many children lack the capacity to handle their upsets, to listen or focus, to rein in impulse, to feel responsible for their work or care about learning, anything that will buttress these skills will help in their education. (Goleman, 1996, p.284)

Goleman suggests that emotional lessons can be built into the fabric of school life through the manner of teachers' engagements with students.

It has been found that pupils hold different conceptions of ability or 'theories of intelligence' which affect how they approach learning tasks and what they hope to gain from them (Dweck and Elliott, 1983). The first theory (the 'entity' theory of intelligence) involves the belief that intelligence is a rather stable, global trait that can be judged to be adequate or inadequate. This trait is displayed in one's performance, and the judgements of that performance can indicate whether one is or is not intelligent. The second theory ('incremental') involves the belief that intellectual competence consists of a repertoire of skills that can be endlessly expanded through one's efforts. While most older children understand both views of intelligence it appears that different

children, independent of their actual ability, tend to favour one or the other. As one would predict, entity theorists prefer tasks that afford opportunities to avoid mistakes and to be judged competent, whereas incremental theorists prefer tasks that afford them opportunities for learning, and they will happily immerse themselves in enquiry ('How can I do it?' 'What will I learn?').

Teachers can influence pupils towards holding an incremental view of intelligence by treating pupils' errors as natural and useful events rather than as evidence of failure, providing opportunities for pupils to engage in problem solving and enquiry, acting as a resource and guide rather than as a judge, and applying flexible and longer-term performance standards which enable progress towards targets to be recognised.

Papert (1980, p.23) argues that learning to program can influence children towards holding an incremental conception:

> many children are held back in their learning because they have a model of learning in which you have either "got it" or "got it wrong." But when you learn to program a computer you almost never get it right the first time. Learning to be a master programmer is learning to become highly skilled at isolating and correcting "bugs," the parts that keep the program from working. The question to ask about the program is not whether it is right or wrong, but if it is fixable. If this way of looking at intellectual products were generalized to how the larger culture thinks about knowledge and its acquisition, we all might be less intimidated by our fears of "being wrong."

Papert contends that the school ethos instils in children a fear of being wrong:

> School teaches that errors are bad; the last thing one wants to do is pore over them, dwell on them, or think about them. The child is glad to take advantage of the computer's ability to erase it all without any trace for anyone to see. The debugging philosophy suggests an opposite attitude. Errors benefit us because they lead us to study what happened, to understand what went wrong, and through understanding, to fix it.
>
> (ibid., p.114)

Whether or not one agrees with Papert's contention (and schools are not only to blame) the strategy of starting from scratch over and over again, whilst it demonstrates admirable persistence, will undoubtedly slow the learner's progress whatever the pursuit.

He asserts that programming generates spontaneously research problems that teachers and pupils can genuinely collaborate in: 'New situations that neither teacher nor learner has seen before come up frequently and so the teacher does not have to pretend to know. Sharing the problem and the experience of

solving it allows a child to learn from an adult not "by doing what teacher says" but "by doing what teacher does".' (ibid., p.115) Thus Papert envisages modelling as a powerful teaching tool to equip children with the strategic knowledge that they need to tackle problems well and efficiently, and he presents a convincing case for some aspects of programming experience making a difference to pupils' conceptions of learning.

2.6.2 *Motivation*

2.6.2.1 *'Awareness' vs. 'willingness'*

Boekaerts (1988) distinguishes between 'awareness', and 'willingness' to use personal resources in the service of a learning task. She gives the following illustration:

> A good illustration of the distinction between "awareness" and "willingness" is that of a student who has learned via a training programme in metacognitive skills that he should double-check on coherence before he answers a comprehension question. Although this student has access to and awareness of the strategy to use, he may still fail to perform well, simply because he considers the "extra" checking to cost too much extra time. (p.230)

She reviews three lines of research evidence which shed light on the relationship between emotion, motivation and learning. On *test anxiety and achievement*, the degree of salience of success and failure within a classroom is responsible for the observed detrimental effect, although this can be partly alleviated by the quality of instruction, i.e. by reducing its density and giving frequent previews and reviews. On *goal vs. ego-orientation*, exam conditions and a competitive atmosphere may elicit appraisals related to such ego-oriented goals as: 'How well can I do the task?' or 'How well can I hide my incompetence?' rather than task-oriented goals such as: 'What can I learn from it?'. Students in task-oriented classes are more likely to believe that putting in effort to gain more skill will help them to succeed in life. On *stress and coping* and a related research line, *social support*, the most representative view is that the way people think about a stressful situation affects how they respond emotionally and how they try to cope. Stressful appraisals become problematic when they reflect a chronic reaction to taxing situations.

2.6.2.2 *'Surface', 'deep' and 'achieving' approaches to learning*

Biggs (1988) views motives and strategies as making coherent packages; students usually choose strategies that are congruent with their motives. These congruent packages define an 'approach' to learning:

A 'surface' approach is an approach that is embedded in pragmatism, in getting by. The student sets out to meet course requirements minimally by limiting the target to essentials that may be reproduced through rote learning. It yields outcomes which are rich in detail but poor in structure.

A 'deep' approach is where the learner is motivated by intrinsic interest in the subject matter, and where he or she sets out to acquire competence and to discover meaning by actively seeking out connections. It yields well-structured outcomes.

An 'achieving' approach is where the motive is ego-enhancement through achieving good grades, and the congruent strategy is organizing time, working space and syllabus coverage in optimal ways. This can be combined with a surface approach (surface achieving) or a deep approach (deep achieving), for example:

> Deep achieving is used by many of the more successful students in high school and university: they search for structure and meaning, and do so while organizing their time and context optimally. (ibid., p.130)

He discusses two successful metacognitively based programmes designed to influence students towards adopting a deep achieving approach. In the first programme students were encouraged to reflect on their work, particularly in those subjects where they were having greatest difficulty, to keep diaries, to self-test, to pair off with other students to monitor and discuss each other's progress, and to discuss in groups. This was supplemented by use of a regular study skills text used as a background resource. In the second programme students were given their own profiles of study habits which became a focus of class discussion, bringing out the possible need for change and how to go about this. This was supplemented by teaching on particular areas of study, such as note-taking and examination technique.

Biggs recommends to teachers three strategies: (1) discourage a surface approach by removing those practices which induce students to bargain for minimal involvement – excessive assessment and workload demands, pointless busy work and pedantic course requirements, authoritarian approaches – all of which result in pressure and cynicism; (2) encourage a deep approach, through arousing students' interest, ensuring active involvement in learning, and encouraging reflection; and (3) develop an achieving approach, through teaching study skills in a metacognitive context.

Entwistle (1987) discusses motivation in terms of students' attributions of success or failure. He observes: 'Most teachers have come across pupils who,

paradoxically, put effort into avoiding effort: they do all they can to avoid carrying out the tasks set by the teacher. What seems to be happening here, is that pupils are trying hard to avoid either tasks they describe as boring or situations from which they anticipate feelings of failure.' (p.5–6). While fear of failure may spur on successful students, it may have the opposite effect on less successful students. External attributions of failure typically refer to bad luck, the difficulty of the task or unfairness on the teacher's part. The theory is that if students can be helped to move from external to internal attributions, that is, if they can be induced to take charge of their own learning, then they are more likely to increase their effort. But what if a student tries harder and still fails? He or she is led to a more painful attribution, that lack of ability is the cause. Entwistle argues that this conclusion is not inevitable, since learning strategies can be modelled and taught. I would add two further points: it is important that all students experience successful learning outcomes (see Nisbet's observation above), and the students' external attributions should not be treated automatically as excuses, thereby letting the teacher off the hook for the quality of the instruction.

2.6.3 *The influence of the social context*

Resnick (1987, p.42) develops the theme of thinking dispositions, which are:

> cultivated by participation in social communities that value thinking and independent judgement. Such communities communicate these values by making available many occasions for such activity and responding encouragingly to expressions of questioning and judgement. The process of learning is further aided when there are many opportunities to observe others engaging in such thinking activities.

Ennis (1987) lists a set of critical thinking dispositions, the majority of which are best practised in a social context, for example: 'Be open minded: a) consider seriously other points of view than one's own ...', and: 'Be sensitive to the feelings, level of knowledge, and degree of sophistication of others.' (p.12) This list is clearly applicable to situations in and out of school (he discusses a jury reaching a verdict to demonstrate the importance of critical thinking dispositions). Many of the examples that Perkins and Salomon (1988; 1989) draw upon when discussing transfer relate to thinking dispositions such as looking on the other side of the case.

Lipman (1987) has developed programmes of Philosophy for Children, each of which uses class discussion around a set text designed to teach the principles of logic or to apply the reasoning skills developed through earlier programmes to specific academic areas. As such, the texts are of a different character from the

'vapid, vacuous reading materials' typical of reading schemes, and 'secondary texts bulging with unappetizing, unpalatable, and indigestible information' (p.156). This short passage (ibid., p.151-2) gives the flavour:

> Now it's *my* turn. I had to wait *so long* for the others to tell their stories!
>
> I'll start by telling you my name. My name is Pixie. Pixie's not my real name. My real name my father and mother gave me. Pixie's the name I gave myself.
>
> How old am I? The same age you are.

Lipman explains how the teacher prompts children's questions by asking what interests them about the passage. What does it mean to take turns? What is it to be real? There is the problem of Pixie's age. Is she speaking only to one person? Does she have many ages, or is she ageless? The teaching method is both non-authoritarian and anti-doctrinal. The teachers role is to, albeit it in a subtle manner, relentlessly feed rationality into the discussion (Lane and Lane, 1986).

Lipman defines attitudes and dispositions as the individual's response to the quality of the social interaction in the group situation:

> In a classroom where there is intellectual cooperation and intellectual criticism, the lively demanding of reasons for opinions and explanations of puzzling events, and the quest for meanings and the exploration of alternatives, children are motivated to wonder, inquire, be critical, be inventive, and care for and love the tools and procedures of inquiry. They acquire – or perhaps reacquire – a need for principles, ideals, reasons and explanations. (Lipman, 1987, p.160)

This provides one example of a programme based upon *discussion* to develop reasoning skills. Other discussion methods involve the use of Socratic dialogues (Brown and Palincsar, 1989) in which there are five main discussion ploys: (a) systematic variation of cases; (b) counter-examples and hypothetical cases; (c) entrapment strategies to lure students into making incorrect predictions or premature formulations; (d) hypothesis identification strategies to force students to specify their working hypothesis; and (e) hypothesis evaluation strategies. Lane and Lane (1986) discuss the need for teachers to be fully trained in collaborative enquiry methods if they are to be used successfully, and for a shift in outlook from the dominant 'authority/knowledge based' paradigm.

Resnick (1987) outlines a range of opportunities that the social setting provides. Modelling by skilled thinkers allows students to become aware of mental

processes that might otherwise remain implicit. 'Thinking aloud' in a social setting allows others to critique and shape one's performance. The social setting may provide a kind of scaffolding for an individual learner's initially limited performance since within the group extreme novices can participate in performing complex tasks and may eventually be able to take over most of the work. The social setting may serve to motivate students by encouraging them to try new, more active approaches, and through this process students may come to think of themselves as capable of engaging in independent thinking and of exercising control over their own learning processes. The public setting lends social status and validation to the disposition to higher order thinking:

> Engaging in higher order thinking with others seems likely to teach students that they have the ability, the permission, and even the obligation to engage in a kind of critical analysis that does not always accept problem formulations as presented or that may challenge an accepted position. (Resnick, 1987, p.41)

The reciprocal teaching method adopted by Palincsar and Brown (see 2.4.2.3) to encourage self-regulated reading incorporates modelling and expert scaffolding within a cooperative learning environment:

> An adult teacher and a group of students take turns leading a discussion on the contents of a section of text that they are jointly trying to understand. The discussions are free ranging, but four strategic activities must be practiced routinely: *questioning, clarifying, summarizing* and *predicting*. ... Throughout, the adult teacher provides guidance and feedback tailored to the needs of the current discussion leader and his or her respondents. (Brown and Palincsar, 1989, p.413)

Collins, Brown and Newman (1989) discuss this approach in terms of how it represents a successful model for *cognitive apprenticeship* (they also use two other examples – Schoenfeld's method for teaching mathematical problem solving and Scardamalia and Bereiter's method for developing writing skills – see 2.4.2.3). Cognitive apprenticeship (see also Hennessy, 1993) is aimed at teaching the processes that experts use to handle complex tasks. The focus of the learning is on cognitive and metacognitive skills and processes, rather than on physical skills and processes as in traditional apprenticeship. Where conceptual and factual knowledge are addressed, their uses in problem solving and carrying out tasks are emphasized; thus conceptual and factual knowledge are exemplified and situated in the contexts of their use (situated learning). The tasks and problems are chosen to demonstrate the power of certain methods, to give students practice in applying them in diverse settings, and to increase the complexity of tasks slowly. The abstract principles underlying the

application of knowledge and skills in different settings should be articulated fully by the teacher.

Cognitive apprenticeship uses learning through guided experience (just as in traditional apprenticeship) but it requires the externalization of processes that are usually carried out internally: 'Cognitive apprenticeship teaching methods are designed to bring these tacit processes into the open, where students can observe, enact, and practice them with help from the teacher and from other students.' (ibid., p.458)

Collins, Brown and Newman identify the methods that are used in cognitive apprenticeship: *modelling; coaching* which is related to specific events or problems that arise as the student attempts to carry out a task; *scaffolding* which refers to the supports the teacher provides to help the student carry out a task; *fading* which refers to the gradual removal of supports until students are on their own; *articulation* is any method of getting students to articulate their knowledge, reasoning or problem-solving processes within a domain; *reflection* enables students to compare their own problem-solving processes with those of others; and *exploration* which pushes students into a mode of problem solving on their own.

A theory which has influenced many *group problem solving* approaches is the *Zone of Proximal Development* (Vygotsky, 1978; Brown and Palincsar, 1989). This theory invites consideration of the child's actual development level (the problem solving that he or she can do unaided), the potential development level (the problem solving that he or she can do under adult guidance or in collaboration with more capable peers), and the zone between the two. Vygotsky argues that what children can do with the assistance of others is more indicative of their mental development than what they can do alone. Through social interaction the child gradually internalizes the newly awakened processes which form the springboard for new learning. This can happen in social settings where there are no explicit instructional goals, such as in informal apprenticeships.

2.7 COMPUTER PROGRAMMING IN SCHOOLS

This section will examine the following closely related issues of justification (should everyone learn programming at school, and what form(s) should programming take?), transfer (what might transfer from programming and how?), understanding what is involved in learning to program, and effective teaching methods.

2.7.1 *Should computer programming be taught in schools, in which form(s), and to whom?*

This question leads quickly into contentious debates amongst computing science educators. Is learning to program an important aspect of cultural literacy in a world where computers are all pervasive, or is it a vocational skill that only a few students need learn?

Some university educators are of the opinion (generally privately held) that schools should not attempt to teach programming. In this view, programming is an elitist activity which is only of interest to advanced students who opt to study a computing science course at university. On the interface between primary and secondary school, and secondary school and further or higher education, one can always find teachers, no matter their subject, who would prefer a 'clean slate' on which to write. Fortunately this position is becoming less and less tenable as attention shifts towards building partnerships between the different sectors of education, articulation of courses, and differentiated teaching.

In an article entitled, 'Should We Teach Students to Program?' Soloway (1993) invites some of the main proponents of schools' computing in America and Canada to put their case. He describes the debate surrounding this issue, which tends to focus on the relative merits of different programming environments, as 'religious wars' (p.24). I shall examine the broad educational thrust of the arguments, avoiding some of the more technical discussion concerning different programming paradigms[9]. (Soloway and Guzdial play devil's advocate below by presenting some of the main arguments for teaching specific applications rather than a general purpose programming language.)

Elliot Soloway and Mark Guzdial (University of Michigan) argue that students need to learn design strategies for specific domains within scaffolded environments, in the manner of the graphics artist learning to use Computer Aided Design software. An example would be a physics student being guided in the design of a microworld to explore physics concepts using a library of

---

9 The British Computer Society A Glossary of Computing Terms Eighth Edition, 1995, uses four classifications of programming languages: *imperative* – the language is used to give instructions on how to solve the problem (an algorithm, or conventional program); *functional* – the language is based on functions applied to sets of data; *logical* – programming involves adding a large number of 'facts' and 'rules' to the system, using mathematical logic notation, and asking the system to deduce conclusions from these; and *object-oriented* – a type of language where the programmer defines not only the type of data, but also the allowable operations that can be carried out on the data.

domain-specific, re-usable components. This approach achieves the goal of empowering students to manipulate, on a routine basis, computational media. The only limiting factors are students' imagination and their mastery of the requisite design skills. In comparison, the learning curve for general purpose programming languages is steep: 'let's be honest, to make the computer truly sing and dance, one needs to write significantly sized programs,' which, they say, is no easy feat (ibid., p.22). Therefore only those who have a vocational interest in computing should learn to use a general purpose programming language.

Michael Clancy and Marcia Linn (University of California, Berkeley) argue that everyone benefits from learning to program so long as it is taught effectively. Programming is ubiquitous in modern society; the powerful foundational models for applications such as spreadsheets, databases and expert systems are to be found in programming. Students should therefore learn to use a powerful general purpose programming language taught through examining case studies of problems. A case study approach makes programming easier to learn, creates 'power users' of applications, and a receptive audience for the technology of the future. 'Such courses also attract talented students who will develop the innovative applications that our complex society continues to demand.' (ibid., p.22) Unlike the kind of environments favoured by Soloway and Guzdial, students explore the high level operators (rather than treating them as black boxes), solve problems in other disciplines as well as in physics (or wherever), learn complex concepts like recursion, and hone their skills in testing and debugging.

Andrea DiSessa (University of California at Berkeley) argues: 'We must have programming, in one form or another, or we will cut off half the power of the medium – like reading without writing.' (ibid., p.23) Creating a computational medium requires three things. First, it must be easier to learn and do than existing programming languages. Second, it requires expressiveness and usefulness: 'Most current languages were simply not designed keeping in mind that people might use them every day, as a medium.' Third, programming needs to climb out of the vocational ghetto: 'Lisp and Pascal are the creations of people interested in doing computer science and instructing new computer scientists ...'. A genuinely computer literate classroom culture should be the main goal of computing educators, one which operates through sharing ideas and artefacts.

Phil Miller (Carnegie Mellon University) sees Soloway and Guzdial's 'applications' view of programming as nurturing student creativity but also as

rather sanitized: 'it certainly doesn't want to get programming all over its shoes by stepping into a pile of it.' (ibid., p.23) He argues that people who program are empowered, and learning to program in the dominant idiom (a general purpose programming language) makes it easy for students to study, understand and extend exciting software artefacts. Programming should be embedded in rich cognitive contexts and should not be taught as an end in itself, which makes it uninteresting and difficult to learn, and also not very useful. It should be fun.

Mitchel Resnick and Seymour Papert (MIT) call for a fundamental rethink of how we introduce programming in schools. They argue that programming has been isolated and disconnected from other subjects, from students' interests, and even from other computer activities, since writing programs and using applications are often viewed as totally separate activities. Instead, they argue, students should explore, experiment and express themselves, for which they need new programming tools. Through programming, students gain new expertise in making things, which, of course, they can also gain through using applications, but, 'programming allows a degree of flexibility that is missing in the fixed menus of standard applications.' (ibid., p.24) Perhaps most importantly, programming can deeply affect students' world view:

> In recent years, programming-based ideas and metaphors have spread through the sciences and social sciences, changing how researchers think about all types of systems. Because of programming-based ideas, entomologists now think about ant colonies differently, cognitive scientists think about the mind differently, management researchers think about organizations differently. When students use our new programming environments, we find that they, too, develop new ways of looking at the world. (ibid., p.24)

Thus the response of this group of authors to the question, 'Should we teach students to program?' is a resounding 'yes', however opinions differ on the form that programming should take – each highlights a particular programming environment as effective – and the main purposes that learning a programming language should serve. Several themes run through the responses such as the need to make programming activity *authentic* to the student and serve a *useful* end, and the need to make programming *easier to learn*.

Yoder and Moursund (1991) explore how learning 'traditional' programming (using a general purpose programming language) interfaces with learning to use applications software. They argue that to teach applications well requires, at some point, the teaching of some programming concepts, since there are a

number of key ideas common to problem solving in a computer environment irrespective of the type of software used. These key ideas include: specifying actions that will be carried out by the computer; procedural thinking; representing data using the computer; and debugging. Thus, in both programming and applications the user *specifies a variety of conditions and actions* to be carried out immediately or stored for use later. Since most recent applications are in some way customizable, and some, such as Hypercard, come with 'full-blown' programming languages, the boundary between doing 'traditional' programming and using applications software has in any case become much fuzzier. The use of *procedures and procedural thinking* is an integral part of programming, since good programming style requires developing programs in subprograms, each of which is designed to carry out a particular procedure. Procedural thinking is also required to solve any complex problem using, for example, an integrated package (applications software which combines word processor, database, graphics, communications etc.). The way in which *data is represented* using the computer must be understood, for example, programming students learn to handle a variety of simple data types, such as integers, real numbers and text strings, but these data types are also to be found in spreadsheets and databases. *Debugging* cuts across both environments. Why has my program produced the wrong output? Why is my spreadsheet giving me incorrect answers? Why can't I get this particular feature of the software to work in this particular case? Communicating expert knowledge of debugging skills to students would help equip them with the necessary skills.

Yoder and Moursund argue that such key ideas should be taught in a manner which will promote their transfer to other programming languages and/or other computer applications. They express concern that programming has been dropped from many of the teacher education programmes designed for those who will specialize in the use of technology in education.

The rationale for programming to form an integral part of all computing studies course provision in Scotland has been examined briefly in the introduction (in 1.2.2). These courses are designed to appeal to a broad range of students and not only to those intending a career in computing. The rationale includes: to enable students to gain an understanding of how the computer manipulates information; to enable students to gain practical experience of solving problems; to provide students with access to the advanced 'programmable' features of applications; and as a preparation for more advanced study. The preamble to the Arrangements for Higher Grade

Computing Studies (SEB, 1988, p.5), presents a broad conceptualization of a computing language:

> One of the key ideas in computing is that of a language that is precisely defined. There is a wide variety of such languages in use and pupils will meet many of them in this course: for example, as command languages of operating software and as conventional high level programming languages. Another suitable context for the development of ideas on languages used in computing is provided by General Purpose Packages. These are significant problem-solving tools in their own right and the unifying themes of the course give the means for evaluation and comparison of the languages associated with the packages.

This analytical view of computing languages is designed to foster transfer to new programming and applications environments (the course provides a range of analytical tools to use). It makes sense to promote the learning of computing languages through a conventional high level programming language and general purpose packages. Whilst a high level programming language lacks the immediate power of an applications language for certain purposes (for example, the language primitives available in a database package, such as sort and search, are much closer to real-life applications, which makes it easier to learn and to use), it has much greater versatility.

This analytical approach contrasts somewhat with the notion of a programming language as an 'expressive medium' (see the earlier references to DiSessa, Miller, Resnick and Papert). When children use, for example, Logo as an expressive medium to explore mathematical ideas or design their own adventure games, they should hardly be aware that they are 'doing programming' at all. This requires a high degree of fluency in the language, since otherwise inability to program will get in the way of the activity in hand. However it would surely be difficult to build this degree of fluency without fostering an analytical approach.

Those who favour an analytical approach must also ensure that students have numerous opportunities to experience at first hand the creative power of a programming language. In this important respect, much of what passes for programming in Scottish secondary schools falls down badly. Programming exercises are frequently dull, routine and disconnected in precisely the manner that Resnick and Papert describe. The experience could be much improved for secondary students if programming courses adopted a more learner centred philosophy (borrowing from the Logo tradition). Programming tasks should be designed to include scope for personal creativity and for the end product to be pleasing to the student, and should not be viewed exclusively as vehicles for

teaching or assessing programming concepts and techniques. This could go some way towards reducing the 'hard edged' technical focus of many computing studies courses, and might attract more girls to the subject.

2.7.2 *Further aspects of the rationale for teaching programming – the wider potentials of programming instruction*

Salomon and Perkins (1987) identify the most widespread argument for teaching programming as its possible impact on generalizable cognitive skills. They comment: 'In general programming is a remarkably rich cognitive enterprise that might yield many different sorts of transfer effects.' (p.154)

They identify six broad and overlapping categories of transfer from programming that might occur:

- Mathematical and geometrical concepts and principles. Some examples are angle and rotation from Logo programming, and variables, algebraic expressions, the real-integer number distinction etc. from numerically oriented programming.

- Problem solving, problem finding, and problem management strategies. Some examples are breaking a problem into parts or relating it to a previously solved problem, planning, and the kind of diagnostic thinking involved in debugging.

- Abilities of formal reasoning and representation. Some examples are thinking of all possible combinations, and constructing mathematical models.

- Models of knowledge, thinking and learning. Some examples are pupils' conceptions of ability (see 2.6.1) which may be influenced towards an incremental conception, procedural thinking, and metacognitive awareness.

- Cognitive styles. Some examples are precision, and reflectivity over impulsivity.

- Enthusiasms and tolerances. Some examples are persistence, and enthusiasm for meaningful academic engagement.

Many of these elements are present in Dromey's (1982, p.1) introduction to computer problem solving:

> Computer problem-solving can be summed up in one word – it is *demanding*. It is an intricate process requiring much thought, careful planning, logical precision, persistence, and attention to detail. At the same time it can be a challenging, exciting, and satisfying experience with considerable room for personal creativity and expression. If computer problem-solving is approached in this spirit then the chances of success are greatly amplified.

Dromey observes that the conscious *depth of understanding* needed to design effective computer algorithms is far greater than we are likely to encounter in most other problem solving situations. His text on computer problem solving discusses a range of general heuristics such as focusing on a particular problem to gain a foothold on the solution of a more general problem, exploiting the similarities amongst problems, assuming that you have a solution and working backwards from it, and using a divide-and-conquer strategy to successively split a large problem into smaller and smaller subproblems until the solution to each can be achieved straightforwardly (top-down design). (Unfortunately, by page 20 the text has become so immersed in the finer intricacies of formal mathematical proof that it has left the average reader behind!)

Still other potentials are present in Papert's (1980, p.60–1) account of the types of procedural thinking involved in doing Turtle geometry (see also 2.6.1):

> One does not need a computer to draw a triangle or a square. Pencil and paper would do. But once these programs have been constructed they become building blocks that enable a child to create hierarchies of knowledge. Powerful intellectual skills are developed in the process – a point that is most clearly made by looking at some projects children have set for themselves after a few sessions with the Turtle. ... Pamela ... began by teaching the computer SQUARE and TRIANGLE... Now she saw that she can build a house by putting the triangle on top of the square. So she tries ... The triangle came out inside the square instead of on top of it!
>
> ... The process of debugging is a normal part of the process of understanding a program. ... There are many ways this bug can be fixed. Pamela found one of them by playing Turtle. By walking along the Turtle's track she saw that the triangle got inside the square because its first turning move in starting the triangle was a right turn...

Because, Papert argues, Turtle geometry encourages the conscious, deliberate use of problem solving strategies (such as the heuristic of 'playing Turtle'), it is a natural aid to learning other things.

In a study with six year-olds, Clements and Gullo (1984, p.1052) have investigated the following wide range of potentials from Turtle geometry (the conduct and outcomes of the study are discussed in the next subsection):

> 1. In Logo programming, children invent, construct and modify their own projects; therefore, Logo programming might facilitate divergent thinking.
> 2. Because Logo is designed to encourage children to reflect on how they think, programming should lead them to develop metacognitive abilities, especially the ability to realize when they do and do not understand instructions.
> 3. Similarly, Logo programming may develop reflectivity in children as they think about their errors and how to correct them.
> 4. If computer programming can allow children to master ideas formerly thought too abstract for their developmental level, it may accelerate cognitive development...
> 5. Finally, because Logo programming involves giving explicit spatial commands, it should increase children's ability to describe directions from their own and others' perspectives.

Linn (1985) stresses the potential of programming instruction to improve problem solving skills. Certain features of the computer learning environment demand very complete and accurate solutions to problems, and students are also highly motivated to use it:

> Programmers must decompose problems into subproblems and then generate sets of step-by-step instructions to solve the subproblems. In addition, the computer learning environment is interactive in that by testing the problem solution, the problem solver can get feedback about how effective the solution is and can use this information to modify the solution. Thus, writing a program requires the student to explicitly use some potentially powerful problem-solving skills. (p.14–5)

However Salomon and Perkins (1987) caution against regarding programming as a, 'kind of cognitive playground', in which, 'mere engagement in the activity of itself would exercise the mind as real playgrounds exercise young bodies, without any need for instruction finely tuned to provoke such consequences.' (p.163) In the absence of such instruction one would predict negative findings on transfer, according to the theories on transfer discussed earlier in 2.4. Predicting transfer from programming is not easy, even when instruction has been designed to foster it:

> While two tasks may both involve a precise cognitive style, breaking problems down, knowledge of geometry, high cognitive load, skills of deductive thinking, or whatever, they may engage these in crucially different ways not captured by such holistic labelling. (ibid., p.156–7)

Only a finer grained analysis will reveal whether the designated skill or ability is likely to transfer. Ideally teachers should assist students to recognize how the skill or ability is engaged differently across a range of situations.

Furthermore, any rich constructivist activity may involve a somewhat similar range of potentials, albeit that programming provides certain opportunities not offered by many other highly constructivist activities, such as the planning and debugging of logical procedures:

> While current research may suggest that programming is *a* way of developing cognitive skills, it is by no means clear which is the *best* way. Indeed one might doubt that there is a best way general across individuals, settings, and cognitive objectives. (ibid., p.164)

These arguments lead towards programming having to justify its place in the curriculum as a valuable activity in its own right, in the kind of terms discussed previously (in 2.7.1). I would argue that it must be taught for understanding and to enable students to develop competence and confidence in their programming abilities. The teaching must also not be narrow. It must stress generic principles and wider applications in order to exploit the full potential of programming to develop cognitive skills and positive affective attitudes towards learning. The focus on understanding and on generalizable aspects should form a synergy.

2.7.3 *The evidence on transfer from programming*

Findings of studies on transfer from programming have been very mixed. Across these studies, a wide range of cognitive skills has been investigated, and the age range of students (from pre-school to undergraduate) varies considerably as does the range of teaching methods and experimental conditions. Such compounding factors make the interpretation of evidence difficult.

Salomon and Perkins (1987) looked for evidence of hugging and bridging to explain the outcomes of transfer studies by Pea, Kurland and colleagues (1984; 1986), Linn (1985) and colleagues, and Clements and Gullo (1984).

Pea and Kurland (1984) conducted two studies on the impact of Logo programming on planning ability with 9 and 12 year olds, using an experimental design. In the first study, thirty-two children either learned Logo programming through self-initiated and self-guided exploration for thirty hours, or belonged to a 'no treatment' group. The pre- and post-test consisted of an elaborate classroom scheduling task. There were no significant differences

in the performances of the two groups. The second study was similar to the first, except that the teacher took a more directive role in guiding the children's Logo explorations and gave immediate feedback on a planning task, encouraging them to notice the similarity to programming. Again the outcome was the same.

Kurland, Pea, Clement and Mawby (1986) conducted a third study, this time with high school students. One class of fifteen students learned programming in BASIC and Logo over two academic years, and their performances in a range of measures (conditional reasoning, exposing flaws and ambiguities in directions, a variant of the classroom scheduling task, a maths test emphasizing variables, and algorithm design and analysis tasks) were compared with those of students who received only one year's instruction or none. Again the findings were negative. It was noted by the researchers that, even after two years computing science instruction, students did not have very good programming skills.

How are these negative findings explained? Salomon and Perkins identify that students had not developed fluency in programming, and received no practice on learning activities that extended beyond the context of programming, therefore the conditions for low road transfer were not met. Also there is no indication of any explicit abstracting and bridging activities, and since these are typically absent in conventional instruction, one can presume that the conditions for high road transfer were not met either.

DiSessa (1987) comments on how the Pea and Kurland findings have been used to criticize Papert's stance on the benefits of programming, however Papert argues that this criticism radically misinterprets his position, since to expect planning skills to develop from children who barely know programming and whose intellectual milieu is otherwise barren of concern for those skills is absurd. This, and other similar discourses, has led DiSessa to call for more theoretical clarity on exactly what kind of higher level knowledge can be developed with programming, how and when it is useful in other activities, and how to support its development with activities outside programming. Singh (1992) criticizes the Pea and Kurland studies on a range of grounds: the full range of planning strategies that the children may have developed cannot be tapped by the classroom scheduling task alone and therefore it provides an insufficient basis for making a statement about all planning, the scheduling task may have been too complex for the children to understand, the children may not have spent long enough learning Logo, and the sample size is too small. He argues that research in Logo would benefit from a basic, in-depth

study of problem solving at the various levels of learning Logo (using repeat commands, writing procedures, combining procedures, using variables, etc.) in order to clarify what is entailed in learning it. Problems arise when researchers without this awareness devise tests, which may result in misleading and invalid results.

Linn (1985) and her colleagues set out to examine the effects of learning to program in BASIC on students' problem solving abilities. She has developed a theory on how problem solving skills can be developed through programming, which draws on research analysing the behaviour of expert and novice programmers, extensive observations of and interviews with students of programming and expert programmers, and successful programming texts. The theory identifies a 'chain of cognitive accomplishment'.

The first link of the chain consists of the *features of the language being studied.* Students typically demonstrate this knowledge by comprehension of already coded programs, for example, they might predict the output from a short program or reformulate a simple program. Knowledge of the language features is necessary for using the language but not sufficient for improving problem solving. Introductory courses often emphasize learning language features rather than how to use them to solve problems.

The second link of the chain consists of *design skills*, or techniques for combining language features to form a program that solves a problem. These include *templates* – stereotypic patterns of code which can be used each time a given task is encountered – and *procedural skills*, such as planning the solution, testing the plan and reformulating the plan if any tests fail. Linn notes: 'Expert programmers use these skills effectively, whereas novices often fail to plan their problem solutions, turn in untested programs, and are unable to fix programs that do not perform the desired task.' (Linn, 1985, p.16) Acquisition of design skills can be assessed by asking students to write programs which require the use of commonly learned templates to solve reasonably complex problems. However often students are not given problems that allow them to use design skills, and they are not encouraged to learn them. Linn explains the value of teaching design skills:

> Students who fail to engage in planning and therefore, fail to decompose a problem into subproblems, set themselves a much more difficult task. Keeping track of all aspects of a problem by writing a large unsegmented program requires much more effort than writing a series of smaller programs and then writing a program to integrate the smaller programs. In addition, planning with a sense of available templates in mind will further reduce the task because some of the smaller programs will already

> be written. Thus, by planning, students can use their knowledge of templates and can make the task less complex. (ibid., p.16)

The third link of the chain consists of *problem-solving skills applicable to a variety of formal systems.* These include both generalized templates and generalized procedural skills. To assess these skills students can be asked to learn a new programming language or to solve problems in another formal system.

Thus, Linn implies, the higher the programming achievement, the more general the scope of the problem-solving skills acquired through programming. To investigate this theory, twenty-four talented students were selected from a sample of some 2400 middle school students learning BASIC. Twenty-one of these students came from two classes where the teaching of programming was considered to be exemplary. In one class the students were explicitly taught templates: 'Programs from students in these classes were easily recognized. These students could use their templates flexibly to design new programs.' (ibid., p.26) In another class, the use of procedural skills was encouraged through, for example, requiring students to attempt to locate 'bugs' in their programs for ten minutes before getting help, and using guided discovery techniques to help students reformulate code when their programs failed to work properly. The twenty-four students received a brief introduction to a new elementary programming language and they then attempted to write three programs in it. Observations revealed that many of the students handled these problems with evident procedural skill and in some cases used templates seemingly generalized from BASIC. Salomon and Perkins identify mindfulness of higher-order problem solving processes as the key feature of the instruction that has fostered near transfer, and speculate that far transfer may also have been possible.

In a complementary study which took place in the naturalistic classroom setting with 600 middle-school computing science students learning BASIC, Linn and colleagues found, after twelve weeks of instruction in 'typical' classrooms, that students were only beginning to master BASIC and were mostly at the language feature link of the chain, however students who received 'exemplary' instruction and who were of a slightly higher ability were at least at the design link of the chain. Linn remarks on the tremendous range of student performance after an introductory course; average performance on the final programming assessment ranged from 17% correct at one typical school to 86% correct at one exemplary school.

A later study by Linn and colleagues (Linn, Sloane and Clancy, 1987) revealed a range of aspects of the instructional setting and of student characteristics that affect the degree of proficiency gained by students on more advanced high school programming courses. These aspects included: students' prior experiences of programming; the level of supervised access to computers in and out of class time (the most significant factor); being required to plan on-line activities; the teacher doing most of the instruction with small groups or in tutoring situations; and the explicitness of the instruction, which may go hand in hand with extensive on-line access since it clarifies what students have to do during on-line access. Overall, the students seemed to benefit from instruction that made the templates needed for solving programming problems explicit, albeit that some students may have invented their own or failed to learn them.

Linn's general conclusion is that, although gains in problem solving skill occur slowly, introductory courses can start students in the right direction by making explicit those aspects of programming that are likely to generalize to other disciplines (Linn, 1985, p.15). However, she warns, the templates that students invent through unguided discovery may not generalize because they involve unrestrained flow-of-control or other unhelpful ideas, also if procedural skills are not made explicit these too may not generalize, and if both are too closely tied to the subject matter neither may generalize (she recommends that students should learn several programming languages to help them to identify the more general forms of these entities). According to Linn, the implications of the study are clear; there must be serious efforts to improve the preparation of teachers for programming classes, programming curricula and resources.

There is support for Linn's conclusions in the findings of a study by Pintrich, Berger and Stemmer (1987) into two 'typical' programming classes. They analysed the programming behaviours of twenty-three moderate to high ability, advanced programming students in two high school Pascal programming classes. The course (followed by both classes) stressed programming methodology, algorithms and data structures. The class met every day for 50 minutes and followed a lecture then individual assignment format. Observations (for two hours per student) on the frequency and timing of nineteen categories of behaviour (run program, enter code, edit code, planning, reformulating code, waiting for help, etc.) revealed that students spent little time on planning their programs or writing their code before they started to key it in. Their basic operating strategy was to, 'get on the machine and start programming.' (p.460) Also their debugging strategy consisted mainly of trial and error: 'In general, they would tinker with the program, making

small changes in syntax or a subroutine ... and then recompiling the program to see if it worked properly.' (ibid., p.460) The researchers noted that there was little direct instruction in design skills and the teaching focused mainly on getting students to learn the language features of Pascal. Stress was laid on the importance of completing projects on time, which encouraged a 'brute force' style of programming, and student-teacher interactions were short and usually focused on hardware problems or specific questions about Pascal language features. There were no formal opportunities for students to have extended discussions.

Clements and Gullo (1984) assessed the effects of learning programming on children's cognitive style (reflectivity, divergent thinking), metacognitive ability, cognitive development and ability to describe directions. The study involved eighteen six year-olds who were randomly assigned to one of two treatment groups, programming in Logo, or computer-assisted instruction (CAI) which served as the control, for twelve weeks. The pre- and post-test involved a wide range of instruments, for example, reflectivity was assessed by the time taken to compare and match pictures of familiar objects and the number of correct matches, and metacognitive ability was assessed through two tasks involving incomplete instructions to see if children realized their lack of comprehension. The methods of instruction for both groups were clearly outlined. They describe a condensed illustrative programming lesson which involves two or three children working at one computer with a teacher:

> Children first planned what they would program the turtle to draw. They drew their pictures with black markers on a piece of paper. Two children drew a house and a flower. They then were encouraged to lay tracing paper over that picture and trace *one* piece of their drawing (e.g. the square that was the bottom of the house), labelling that "1." They used a clean sheet of tracing paper to trace another piece (the triangular roof), labelling that "2," and so on. These pieces were then defined as procedures that were typed into the computer with one or two special support programs (written by one of the researchers) that allowed the children to define a procedure and simultaneously watch it being executed, editing it at any time that was necessary. ... If a procedure was not doing what the children had anticipated, they were encouraged to think it through: "What did you *tell* the turtle to do? What *did* it do? What did you *want* it to do? How could you change your procedure?" and so on. Finally they assembled the procedures in a superprocedure they might name HOUSE, which uses the SQUARE procedure to draw a square, moves into position, and uses the TRIANGLE procedure to draw a triangular roof.
>
> (Clements and Gullo, 1984, p.1054)

The lessons were also carefully sequenced. Children learned to: move the turtle and draw lines using single keystrokes (two sessions), write procedures

to draw shapes using single keystrokes (four sessions), combine procedures into superprocedures to draw more complex pictures (four sessions), do the same in the full language (eight sessions), and plan a superprocedure first then sub-divide this into subprocedures (top-down planning – six sessions).

The findings from this study on all measures except one (cognitive development) were positive, demonstrating far transfer from programming. Clements and Gullo suggest that perhaps the children were not yet ready to deal with abstract constructs, as an explanation for there being no significant effect on their cognitive development.

Salomon and Perkins identify that the strong scaffolding provided by teachers, aided by the student-teacher ratio, would help students to reflect upon and thereby abstract from their programming experiences (a form of bridging). They also discuss why transfer did not occur for the children in the CAI treatment group. These children also worked in small groups at one computer with a teacher, who led them in occasional discussions and prompted them through guided questioning to make their thinking and mastery of concepts explicit. They conclude that CAI provided much less to abstract *from*. Clements and Gullo's (1984, p.1056) discussions of positive findings on a range of measures make the differences clear: 'Children in the Logo group may have increased their ability to produce original ideas and to produce creative ideas ... because the Logo programming facilitated divergent thinking within a figural context', whereas CAI treatment, 'often emphasized convergent rather then divergent thinking (e.g. timed presentations of addition exercises)...'. Also, 'The nature of programming in Logo necessitates thoughtful advanced planning, reflection on one's thinking, and explicit analysis of errors in "debugging"...', whereas the CAI treatment, 'included instructional programs that rewarded quick responses.' Finally, 'The ability to monitor one's own thinking and realize when one does not understand may also be positively affected by computer programming environments in which problems and solution processes are brought to an explicit level of awareness and in which consequent modification of problem solutions is emphasized. Through consistent feedback in the form of a visual representation of the procedures and sequences of their own thinking processes children may have learned how to monitor those processes.'

It is clear that there are many successful ingredients of this teaching approach beyond the strong bridging that Salomon and Perkins identify. There is the careful sequencing of lessons, collaborative learning in small groups, direct teaching on problem solving strategies, coaching, scaffolding, fading, reflection,

articulation, exploration – indeed, most of the methods associated with cognitive apprenticeship.

On the basis of the overall evidence, Salomon and Perkins reach a number of conclusions. Transfer by the low road from programming seems unlikely: 'One has to have a skill to automize it.' (p.164) Meeting the condition of varied practice to near automaticity requires a time commitment that is out of the question in most educational settings. Therefore, if transfer is to occur from introductory programming courses, then the high road must be 'forced' by instruction that, 'directly and vigorously helps students to think about programming at an abstract level, in terms of the generic strategies involved.' (p.163) However one might question the durability of the effects from short treatments (some are as short as twelve hours), and also their generalizability, noting the differences between treatment effects under well-controlled experimental versus real-life conditions. Such mediated instruction places considerable demands on schools' resources (e.g. it benefits from a high student-teacher ratio) and teachers' skills. However carefully designed curricula, improved programmes of teacher training and the like may make possible in the longer term the routine implementation of programming instruction that will impact on cognitive skills.

There continues to be a steady flow of articles reporting on research on the cognitive effects of learning to program, the majority of which use quasi-experimental designs (see Baron, 1987). Some articles report positive findings on transfer of problem solving skills, such as two studies of a similar design by Palumbo and Reed (1991) and Choi and Repman (1993)[10]. Students learned structured programming techniques over a full semester (75 hours and 100 hours respectively), in which the programming curriculum paralleled Linn's chain of cognitive accomplishments:

> Care was taken to initially develop a strong base in the declarative knowledge required to effectively write programs using the BASIC language. Once this declarative knowledge base was established, students were given the opportunity to develop programs to solve specific programming problems that required use of not only the declarative knowledge base, but also the procedural knowledge as they linked program commands together to form effectively and efficiently coded BASIC programs. ... The curriculum continued in this manner, slowly

---

10 The Palumbo and Reed study focused on a high school level BASIC programming course. The Choi and Repman study focused on two college level programming courses in Pascal and FORTRAN. The problem solving measures were drawn from sections of two general tests of critical thinking – the Ross Test of Higher Cognitive Processes and the Watson-Glaser Critical Thinking Appraisal.

> adding to the declarative knowledge base and then allowing students time, through structured activities and assignments, for the procedural knowledge base to be equally increased.
>
> (Palumbo and Reed, 1991, p.364–5)

A study by Ennis (1994) contrasts two methods of teaching BASIC programming to high school students. All students were taught structured programming. In addition, half of the 38 students (the treatment group) were taught a generalized planning procedure – list the steps they planned to take to accomplish their programming goal, identify and define what was needed, determine what was given, and decide what programming commands they needed to use – while the other half (the control group) received example oriented instruction. Although no statistically significant differences in the general problem-solving abilities of the two groups were found, some interesting qualitative observations were noted. Careful analysis and planning occurred among students in the treatment group because they had a game plan. They spent more of their time thinking critically and they needed to ask fewer questions involving logic. The treatment group often worked in pairs and moved around the classroom environment freely exploring, comparing and devising plans, using their classmates as a resource to formulate their solutions, whereas the control group quietly pursued their own programming goals. The treatment group produced diverse and creative programming solutions whereas the control group produced carbon copies of the example programs. Therefore problem-solving instruction has served as a catalyst to help students analyse, evaluate, interact, make wise use of available resources, and produce solutions in which they express their own ideas.

In a review of Logo instruction designed to promote transfer of learning, Keller (1990) highlights the importance of structure, mediated instruction and the effective use of peer interaction. She reports various teachers' experiences of unstructured Logo environments. In one study the students' programming involved, 'little planning or reflection, much trial-and-error, frequent goal changes when encountering errors, and lack of curiosity as to the nature of the phenomena and bugs appearing on the screen.' (p.58) In another study, the students had, 'trouble setting realistic project goals, did not know how to take the next step in solving their problems, and seemed to accept the logic of the computer on its own grounds rather than struggling to understand it.' (p.58) As the teachers gradually introduced more structure to overcome these difficulties, the students' interest and proficiency increased. Mediated instruction, 'strives to make problem solving strategies clear to the students and helps them to apply the strategies in other situations.' (p.68) Studies

showed a remarkable consensus on the effectiveness of mediated Logo instruction to promote transfer by the high road.

Liao and Bright (1991) have synthesized, using statistical techniques, the results of sixty-five classroom based studies, each of which assessed the relationship between computer programming and general cognitive skills using quantitative comparisons between treatment and control groups. Their main conclusion is that programming does provide a mildly effective approach to developing students' cognitive skills in a classroom setting. Larger treatment effects were associated with: learning Logo and Pascal, which provide support for structured programming; studies of less than three months duration; and studies which do not take place in high school settings. Therefore, they conclude, semester long high school BASIC courses may need to be rethought, and some attention paid to devising instructional methods which will successfully retain students' positive attitudes towards learning cognitive skills.

However Goodyear (1987), in an article which examines the sources of difficulty in assessing the cognitive effects of learning to program, expresses considerable scepticism about the value of controlled experimental studies such as those reviewed by Liao and Bright. He argues:

> A common weakness of empiricist research designs, which offer an educational 'treatment' to an experimental group and then assess performance on one or more post-tests, is the lack of a theoretical account which satisfactorily links 'treatment' and 'effect'. Without such an account, suitable post-tests are difficult to select (or design) and the effects they measure are impossible to explain. It is difficult to see, *a priori*, why involvement in 10 hours of work with turtle graphics should lead to change on questionnaire X or improved performance in examination Y.'
>
> (Goodyear, 1987, p.216)

While an analysis of the cognitive demands and implicit benefits of computer programming might identify the need for planning, precision, the generation and testing of hypothesis, and so on, if the novice programmer's actual experience is radically different from this, what justification is there for hypothesizing specific transfer effects? Only when we understand better what is involved in learning programming can we begin to assess the wider applicability of the skills so learned, and this means that learners must be given time to learn the rudimentary skills, otherwise: 'we will continue to look for differences between "programmers" and "non-programmers" when our programmers can't actually program.' (ibid., p. 218) If learners possess the

rudimentary skills of programming this opens up the possibility of transfer occurring by the low road.

Goodyear sets out four requirements: to investigate what actually occurs when novices are learning to program; to refine our conception of 'transfer of learning' and our understanding of the circumstances in which it is likely to occur; to hypothesize and test for some beneficial side-effects of engagement in learning programming; and only then to make an informed judgement about the place of (non-vocational) programming in the curriculum.

Many other researchers (e.g. Papert, 1980; Cope and Walsh,1990; Thomas and Upah, 1996) have warned of the dangers of reaching premature judgements on the place of programming in the curriculum on the basis of the collected findings on transfer. In addition to criticisms of flawed experimental designs and the narrow scope of many studies, they argue that much of the evidence was collected at a time when programming was new to schools and there was little experience of teaching or researching into it, also the rapid pace of developments in the field of computing has resulted in improvements in programming environments, and research findings on learning in other areas suggest promising new directions.

2.7.4 *Understanding what is involved in learning programming and consideration of effective teaching methods*

The main focuses in this sub-section are the nature of the difficulties that students experience when learning programming for the first time, and the design of instructional settings to enhance their learning of programming. Theories on instructional methods have been influenced by findings on how the knowledge base of expert programmers is organized. A commentary on the prospects for transfer of general cognitive skills from programming continues through the discussion.

As noted previously, a common finding of studies seeking transfer from programming is that the students have not learned the programming skills very well (Perkins and Salomon, 1988). Perkins and Martin (1986) have investigated why this should be. Is it a consequence of inadequate knowledge or neglect of general problem-solving strategies? This question is of relevance to both the pedagogy of programming and to exploring how learning to program might impact on learners' general cognitive skills. If, they argue, there is no strategic shortfall then programming cannot be expected to boost learners' cognitive skills until it moves beyond the fundamentals. On the other hand, if learners' knowledge of the details of the language is adequate, then

programming could provide a natural training ground for the development of general cognitive skills.

Perkins and Martin recognize that this question over-simplifies the issue in two ways: perhaps both are implicated, and knowledge of a programming language is not an 'all' or 'nothing' affair. There may be missing or partly missing knowledge (not retained or never learned), inert knowledge (not retrieved – a problem of transfer), misplaced knowledge (a problem of negative transfer), or conglomerated knowledge (a jamming together of disparate elements in a syntactically or semantically anomalous way in an attempt to provide the computer with the information it needs). These four senses together constitute *fragile knowledge.*

In order to highlight the interaction between strategies and fragile knowledge, Perkins and Martin conducted a series of clinical case studies with twenty high school students taking a first year BASIC course, taught in a careful and conscientious manner by a teacher who was knowledgeable about the BASIC language. The procedure involved presenting each student with a choice of eight increasingly difficult but related programming problems on FOR–NEXT loops. Each student selected a problem that seemed manageable, attempted it in interaction with the experimenter, completed it, moved on to the next problem, and so on, until the end of the 45 minute session. The experimenter watched and asked occasional questions to track the student's thinking. If the student encountered a significant difficulty the experimenter intervened, asking questions and providing information to help the student overcome the problem. The first questions (prompts) were high level strategic questions (e.g. 'What's the first thing you need to tell the computer to do?'). Some prompts were phrased generally and could be used in any problem solving situation, while others were particularized to mention some element of the programming situation. If a couple of prompts did not help, the experimenter resorted to 'hints' which reflected the experimenter's understanding of the solution. If a couple of hints did not provoke progress, the experimenter provided an exact solution (an answer and explanation) to the immediate dilemma so that the student could get on with the rest of the program (provides). Occasionally the experimenter moved directly to hints or provides if the general performance of the student and spiralling frustration signalled that more direct help was needed. This escalation, from prompts to hints to provides, served as a probe of the student's level of mastery and understanding. Each interaction resulting from a student's inadequate handling of a problem was recorded and codified.

Missing knowledge was implicated whenever a prompt or hint failed to resolve a problem, or when a provide at a level above that of writing a line of code was not carried through by the student. About one-third of the recorded episodes fell into this category.

Inert knowledge was implicated whenever a prompt or hint successfully resolved a problem, or when a provide at a level above that of writing a line of code was successfully carried through by the student. Almost a half of the recorded episodes fell into this category. As an explanation of this phenomenon, the authors note: 'In general, one should recognize that an experienced programmer will have a rich network of associations linking various commands and programming plans; if retrieval does not succeed by one route, it will probably succeed by another. In contrast, the network of connections in the novice inevitably is sparse; if retrieval by one route fails for whatever reason, there may be no other ready way.' (ibid., p.220)

Misplaced and conglomerated knowledge together accounted for the remaining 17% of episodes. The main causes were: overgeneralization; recency of learning (various intrusions cropped up as a function of what students were learning in class); taking a stab at the answer, in the absence of any proper recourse; failure to 'close track' with precision (reading back expressions in a programming language to discern exactly what they tell the computer to do); and treating the programming language as, 'much looser, less restrictive, more expressive and more like a natural language than in fact it is.' (ibid., p.223)

Prompts were effective at least a third of the time, and prompts and hints together about half of the time, indicating, 'that students had considerably more knowledge resources than they accessed without help and that much of the time they might help themselves to access that knowledge by self-prompting.' (ibid., p.223) The prompt to close-track occupies a pivotal position in programming, as the following example illustrates:

> But even before running the program she paused, pondered her program for a moment, and realized, "It's going to end up like a square." Naomi's inclination and ability to read back her program and forecast what it would do, rather than presuming it would do as she intended, enabled her to reject plans and seek other alternatives. As noted, this was relatively infrequent among the students.

About half of the time when students close-tracked they succeeded in forecasting a problem or explaining a bug.

Perkins and Martin summarize as follows. Novices' difficulties with programming can be characterized by fragile knowledge exacerbated by a shortfall in elementary problem-solving strategies. They suggest that normally responsible and knowledgeable instruction does not suffice to give students a reasonable mastery of programming:

> Consider for a moment what a challenge programming is. Unlike most school subjects, programming is problem-solving intensive. One cannot even come close to getting by just by knowing answers. Moreover, as the phenomena of misplaced and conglomerated knowledge demonstrate, a freewheeling manipulation of one's knowledge will not suffice either ... The demands of a computer that cannot discern what a program means are inevitably more stringent than the demands of a reader who not only can see through to meanings but may appreciate exploratory and playful extrapolations and stretchings of concepts. Moreover, programming calls for extraordinary perfection. If you get 90% of the words right on a spelling test, you score a 90 ... But if you get only 90% of your commands right in a program, it will not do anything like what it is supposed to. Moreover, the remaining 10% may well introduce several interacting errors that make tracking down the bugs a demanding and frustrating task. (ibid., p.226)

They recommend to teachers three broad approaches. Firstly, highlight the functional role of commands in their generality in order to work against inert knowledge and convey an understanding of exactly what commands do. Secondly, teach programming so as to preserve the exploratory use of the language. A good principle is, 'When in doubt, take a stab at a solution; but check your stab by close tracking.' (ibid., p.227) Thirdly, encourage elementary problem-solving strategies.

They conclude that general problem-solving strategies can contribute from early on to learning a programming language. Postponing the introduction of problem-solving until after the basics of the computer language in question have been learned is therefore not a sensible strategy. This challenges Linn's theory of a chain of cognitive accomplishments in which programming specific versions of problem-solving skills do not come into play until the second link of the chain. The findings of this study also challenge Linn's theory (see chapter 8). In terms of the prospects for transfer of general cognitive skills from programming, they summarize the position as follows: although computer programming provides a natural arena for the development of these skills, it is not an activity whose pursuit automatically develops them.

I would argue that Perkins and Martin present a realistic but not over-pessimistic view of the prospects for novice programmers to gain a reasonable degree of competence in programming (with appropriate teaching) and for

transfer of cognitive skills from programming. The rich and varied examples that the study provides of students' programming difficulties are very recognizable to any experienced teacher of programming, and their detailed analysis helps to make sense of the patterns. Their conclusions do not set out any detailed prescription for teaching programming, which would in any case be unhelpful given the range of circumstances that one finds in introductory programming classes and the need to accommodate the preferred learning styles of students and the preferred teaching styles of teachers.

Papert (1994) has investigated the learning styles of children within Logo programming environments in which they are given sufficient access to computers and freedom to develop a personal style of programming. He uses the term *bricoleur* to describe a method of working in which the child feels his or her way forward, not by following an exact plan, but by a process of negotiation and improvisation. The child does have a goal which he or she is committed to achieving, however the goal is allowed to evolve as the work progresses. Papert provides a short vignette to illustrate the differences between this approach and a 'top-down', divide-and-conquer strategy:

> Jeff is the author of one of the first space-shuttle programs. ... There will be a rocket, boosters, a trip through the stars, a landing. He conceives the program globally; then he breaks it up into manageable pieces. ...
>
> Kevin too is making a space scene. But the way he goes about it is not at all like Jeff's approach. Jeff doesn't care too much about the detail of the form of his rocket ship; what is important is getting a complex system to work together as a whole. But Kevin cares more about the aesthetics of the graphics. He spends a lot of time on the shape of his rocket... He works without plan, experimenting... He frequently stands back to inspect his work, looking at it from different angles, finally settling on a red shape against a black night – a streamlined futuristic design. ... Jeff happens to pass Kevin's machine on the way to lunch and automatically checks out its screen... Nothing new there, nothing technically different, just a red blob...
>
> By the next day Kevin has a rocket with red fire at the bottom. "Now I guess I should make it move..." (p.147–8)

Kafai (1993) created a learning and research environment in which primary aged children, in a computer rich environment, were given freedom to develop a personal style of programming as they created, over a six month period, computer games software in Logo around a fractions theme. She found that most pupils preferred to operate in both styles, sometimes planning ahead and sometimes being guided by the flow of their ideas, depending on their needs at the time (p.330).

I would argue that many introductory programming courses in secondary schools do not provide the conditions or encouragement for students to develop their own programming styles, since they demand a rigid adherence to a software engineering approach (which would suit Jeff but not Kevin). Linn, Sloane and Clancy (1987) stress the need to balance explicit teaching on problem-solving strategies with opportunities for reflection and self-guided learning, since otherwise students may fail to develop self-monitoring skills and autonomous problem-solving abilities. However, sometimes the balance is too far in the opposite direction: 'Teachers who expect introductory students to seek solutions through independent inquiry are dooming their students to frustration unless they find ways to impart the needed independent problem-solving skills.' (Husic, Linn and Sloane, 1989, p.579)

I shall now return to examine in rather more detail what explicit instruction in problem solving within programming might entail.

Linn and Clancy (1992) refer to programming-specific versions of problem-solving skills as design skills, which include *templates* – stereotypic patterns of code which can be used each time a given task is encountered – and *procedural skills* – designing, implementing, testing, debugging and self-monitoring. They have developed a case study approach and a range of case studies (Clancy and Linn, 1992) to teach design skills. Examples of templates which are introduced and elaborated through the case studies are: prompt then read, input – process – output, do something a specified number of times, select from alternatives, input and process until done, accumulate values until done, and search a one dimensional array. In addition to the program code, a template includes an English explanation, pseudocode, a diagram, a 'real-life' analogy, things to look out for when testing and debugging, and links to other templates. Each case study assumes knowledge of certain language constructs and introduces others.

Starting from a statement of the programming problem, a case study includes a narrative description of the process used by an expert to solve the problem written so that a student can understand it, a listing of the expert's code, study questions – *stop and help* solve the problem, *stop and predict* what the experts will do, *stop and consider* alternatives – and test questions. Test questions are designed to assess understanding of the solution and may involve modification of the solution, application to a related context, analysis to compare and evaluate different solutions or approaches to a particular problem, testing, debugging, or reflection to help students identify helpful and unproductive programming practices. It is recommended that solutions should be explored jointly by programming teams. Thus the case study

approach involves explicit teaching on a range of design skills and language constructs, modelling, application and feedback, reflection and collaboration.

The case studies enshrine eight design principles which focus on different aspects of template knowledge and procedural skills. Four are reproduced below:

> **The Recycling Principle**
> Reuse ideas, specifically templates for design, that have worked previously. Elaborate on a known template to solve a new problem rather than attempting to design new code from scratch for every problem encountered.
>
> **The Alternative Paths Principle**
> Generate and evaluate alternative designs for programming problems. Develop processes for finding alternatives and criteria for selecting among alternatives.
>
> **The Reflection Principle**
> Reflect on alternative designs for programming problems and on problem-solving processes. Develop skill in monitoring progress and in diagnosing unproductive activities. Anticipate personal weaknesses and guard against them.
>
> **The Fingerprint Principle**
> Develop effective debugging skills by associating symptoms of bugs (their "fingerprints") with the appropriate bug. Knowledge of common bugs and their associated fingerprints – symptom-bug associations – facilitates debugging.
>
> (Linn and Clancy, 1992, p.128)

The other four case study principles relate to developing multiple representations of design templates (natural language, diagrams, dynamic illustrations etc.), using divide-and-conquer to reduce the cognitive demands of programming, adopting a 'persecution complex' when testing programs, and producing self-documenting code that is easy to understand, modify and debug.

Linn and Clancy provide justification for the case study method by drawing on the wider literature on teaching design skills (e.g. Schoenfeld's work on mathematical problem solving) and research which reveals the deficiencies in conventional methods of teaching introductory programming. For example, students often attempt to organize their knowledge of programming in terms of the language syntax because this is how the information has been presented to them (Linn and Clancy note that in pre-college settings teachers of programming may not know much more about programming than their students). This results in an unwieldy and fragmented knowledge organization, and as a result students have great difficulty with assembling algorithms and often engage in random trial and error to design a program. Also textbooks may fail to provide discussions of the decisions required to design and develop programs, focusing only on the end-product. Modelling of the solution process by the teacher has been found to be very effective:

> Thus, teachers who talk through how they solve problems and include in their discussion some reinterpretation of the problem statement, some backtracking from initial conjectures, and some clear examples of how they use prior problem solutions to solve new problems, are more effective than those who provide a tightly knit presentation of the subject matter but fail to communicate how the subject matter can be applied to a problem. (ibid., p.127)

Evaluation of case studies with high school students learning Pascal programming has revealed their effectiveness for communicating the complex decision making necessary for designing good programs, however there are some limitations, one of which is the readability and length of the expert commentary.

The idea of a programming template has its origins in research into how the knowledge base of expert programmers is organized. It has been found that expert programmers possess a large knowledge base of domain-specific patterns (chunk like configurations or schemata). Soloway and Ehrlich (1984) identify (on the basis of their empirical studies) that expert programmers have and use two types of programming knowledge – plans (equivalent to templates), and rules of programming discourse which capture the conventions in programming and govern the composition of the plans into programs. *Plans* are associated with *goals*, in the manner illustrated below:

> An average requires that a sum of the input numbers be taken, and that this sum must be divided by a count of the numbers summed. These two requirements are *goals* of the problem. In developing a programming solution for this problem, two additional goals (at least) must be put forth: The numbers must be input, and the average must be output. Expert programmers have developed standard techniques for realizing these goals since they are common and appear in many problems. ... In our terminology, such canned solutions are *plans*. (Soloway, 1986, p.851)

When decomposing problems into subproblems, the objective is to identify in the given problem statement various goals for which programming plans have already been developed (this is the macro-level). Plans are then composed (at the micro-level) to form a program. Four strategies for composing plans have been identified – abutment, in which two plans are glued together back to front in sequence (as illustrated in the average example), nesting, in which one plan is completely surrounded by another plan, merging, in which at least two plans are interleaved, and tailoring, in which an existing plan is modified to fit the particular needs of the situation. (All of these methods are discussed in chapter 6.)

However plan composition is rather difficult and error-prone, as the findings of an empirical study by Spohrer and Soloway (1986) reveal. Although novice programmers' misconceptions about language constructs were the cause of some bugs, others arose as a result of plan composition problems. They have devised a (tentative) taxonomy of construct-based problems and plan composition problems. The three construct-based problem types are: *natural-language* – many programming language constructs are named after related natural language words (e.g. and, or) and novices may become confused about the semantics of the constructs; *human interpreter* – novices know how they intend a construct to be interpreted and may assume that the computer will arrive at the same interpretation; *inconsistency* – because novices understand how a construct works in one situation they may assume that it works in the same manner in another, slightly different situation. Two examples of plan composition problems are: *specialization* – novices develop abstract plans that must be customized for a particular new situation, and sometimes a particular abstract plan is inappropriate or not instantiated correctly; *cognitive load problem* – small but important parts of plans may be dropped out or plan interactions may be overlooked.

2.7.5 *Discussion on the pedagogy of programming*

There are many interesting contrasts in the methods of teaching programming proposed by, for example, Papert and Soloway, who seem to be at opposite extremes in terms of the degree of structure that they would seek to impose on students' programming experiences. Soloway recommends teaching a whole new metalanguage of goals, plans and plan composition, identifying each separate plan by its acronym (e.g. the RUNNING TOTAL LOOP PLAN, the COUNTER LOOP PLAN) and placing it in a library. The suggested approach seems over-mechanistic, not practical for larger programs, and requires students to learn an extra language that bears all the hallmarks of the 'hard' computer scientist at work! While this approach may succeed in automating the task of program production in certain tightly constrained circumstances, there are many missing ingredients (such as developing self-regulation). The main fault lies in the assumption that the expert's tacit knowledge can simply be communicated to beginners without allowing them opportunities to construct their own understandings, as though the 'messy' stage at the beginning can simply be bypassed. Goodyear (1987) highlights the opportunity that the literature on expertise creates for confusing the attributes and cognitive demands of the expert programmers' task with the characteristics of the programming undertaken by school students.

Nevertheless, Soloway's analysis adds further depth to our understanding of programming in a variety of ways that are helpful. It provides a finer grained analysis of novice mistakes, a way of assessing the complexity of programming problems in terms of the difficulties of synthesizing plans, and a heightened awareness of the difficulties facing novice programmers when they attempt to decompose problems with only a limited knowledge of programming goals and plans. These aspects can be taken on board when planning a programming curriculum to ensure a reasonable gradation of difficulty in the tasks and sufficient scaffolding to support the learners' early attempts.

The research design used by Perkins and Martin suggests a useful strategy for pedagogical interactions that could be adapted to the classroom setting, and in addition the findings demonstrate how the application of simple problem solving strategies can help compensate for the fragile knowledge of the beginner programmer. Not only does this help students to learn to program better, it also increases the likelihood of transfer from programming. However their caveat that learning to program does not automatically develop cognitive skills must be heeded. (Soloway (1986, p.850) asks, 'Why *should* learning where to put a semicolon in Pascal lead to enhanced problem-solving ability?')

In this regard, Linn's point about avoiding a syntax-driven approach to teaching programming is absolutely crucial. In my work with student teachers I draw on examples from course materials in use in local schools to highlight the many drawbacks of this approach. From the starting point of the general programming specification (see 1.2.4.3) teachers devise a scheme of work in which the language constructs are arranged into some notional order of difficulty. Thus lesson 1 – using the PRINT command to output text; lesson 2 – introducing numeric variables and assignment, and so on. Each lesson places the spotlight on the new language feature by the teacher composing a short program at the front with the class's participation ('What goes in the next line?' 'What does this command do?' etc.). For example,

```
100 // the for.. next loop
110 FOR counter:= 1 TO 10 DO
120   PRINT counter
130 NEXT counter
140 END
```

Pupils rush to computers, type in the program, write down the result (a column of figures from 1 to 10) and then work through some similar exercises until the end of the period (some exercises may require modifying the program or writing a short program). The next lesson sees the introduction of a new language feature:

```
100 // the if .. then statement
110 ...
```

To begin with, most student teachers view this as a perfectly sound approach (it mirrors how many of them were taught programming or mathematics – they have short memories when it comes to their own struggles to learn!). It guarantees coverage and lends a structure to their teaching. The drawbacks are only apparent when it becomes obvious that the children cannot program. This moment arrives when the first coursework assessment is tackled, which requires the analysis of a problem description, and the design, implementation and evaluation of the solution – elements which have been largely absent in the instruction. Furthermore, pupils have no experience of synthesizing elements of a solution since the programming exercises focused on one language feature at a time, they have never needed to worry about when and where each language feature comes into play since this has been cued by the instruction, and they have no bank of previously solved problems on which to draw. These are not minor problems that can be easily remedied, for example, by doing a worked example to demonstrate the techniques once. The shortfall in pupils' programming knowledge is by this time substantial. Some teachers continue to use these methods because they attribute poor performance to certain pupils' lack of general ability, motivation or maturity (for example, they might argue for programming to be taught only at credit level at Standard Grade or to S4 rather than S3 pupils), or to the inadequacies of the programming environment.

The research studies conducted by Linn and her colleagues place an in-depth focus on pedagogical issues and reflect many of the different facets of classroom life. This is in marked contrast to the numerous studies on transfer from programming which provide no useful information on the instructional setting. The underlying instructional theory (the chain of cognitive accomplishments) attempts a separation between declarative knowledge (learning the language features) and procedural knowledge (learning design skills). There seems to be a real contradiction between the case study approach in which design skills are thoroughly integrated, and the suggestion that introductory courses should focus on teaching the language features as the first link in the chain. (I have been unable to resolve this contradiction in her work.)

At the end of the next chapter the characteristics of the programming instruction within the resources utilized by the case study class are outlined.

3 THE PROGRAMMING PROJECT

3.1 CHAPTER OVERVIEW

This chapter examines the development of the programming project and its outputs. It provides an important background to the case study which is the focus of the remaining chapters of the thesis. The chapter begins by an examination of the manner in which the project engaged teachers in collaborative action research to create an effective learning environment for pupils (section 3.2). It then focuses on the nature of the outputs – the learning resources (section 3.3). Chapter 1 has already set out the circumstances surrounding the establishment of the project and some of the characteristics of the learning environment that we sought to create for pupils.

3.2 THE MANNER IN WHICH THE PROJECT ENGAGED TEACHERS IN COLLABORATIVE ACTION RESEARCH

Three key aspects of the project are examined in this section – its central philosophy (the 'learning teacher'), the nature of project activities, processes and evaluation, and the types of dilemmas facing teachers undertaking research in their classrooms.

3.2.1 The 'learning teacher'

The notion of the 'learning teacher' was a central philosophy of the project, and is encapsulated by Stenhouse (1980, p.244):

> The improvement of teaching is not the linear process of the pursuit of obvious goals. It is about the growth of understanding and skill of teachers which constitute their resource in meeting new situations which make old aspirations inappropriate or unattainable.

and by Fullan (1993, p.138):

> It seems obvious to say, except that we don't practice it, that you can't have a learning society without learning students, and you can't have learning students without learning teachers. Inner learning (intrapersonal sense-making) and outer learning (relating and collaborating with others) run together, but they are also separable.

Time for 'inner learning' and opportunities for 'outer learning' were therefore important considerations, with the concomitant need to pace the project well, and to secure continued funding on an annual basis and teachers' release from school to participate in project events on an on-going basis.

The research aims directed towards teacher learning can be stated as involving a disciplined enquiry into learning and teaching in programming classes, some experimentation (within an agreed framework), and a sharing of expertise to develop, implement and evaluate the learning resources.

Stenhouse (1983) views the object of educational research as, 'to develop thoughtful reflection in order to strengthen the professional judgement of teachers,' (p.192) but he also recognizes that the natural cry from the field is, 'for the reassurance of certainty to ameliorate the agony of responsibility.' (p.193) This tension was very evident in the demands from some quarters for 'answers', mostly from people who chose to remain on the periphery but felt that the funding that had gone into the project justified their demands. The recurrent theme was how to get 'credit' pupils through the programming content in forty hours. In order to channel more resources towards this issue and to stimulate a deeper level of debate, a proposal was submitted to the Scottish Office Education Department (SOED) for small-scale funding to investigate, within some project schools, methods of accelerating the progress of able pupils. This proposal was successful and resulted in the publication of two reports (Kirkwood, 1996; 1997a).

The programming project came to be construed by the participants as an 'action research' project because of its intentions, which can be recognized in Elliott's (1991) descriptions of action research:

> A felt need, on the part of practitioners to initiate change, to innovate, is a necessary precondition of action research. It is the feeling that some aspect(s) of a practice need to be changed if its aims and values are to be more fully realized, which activates this form of inquiry and reflection. (p.53)

and:

> Action research improves practice by developing the practitioner's capacity for discrimination and judgement in particular, complex, human situations. It unifies inquiry, the improvement of performance and the development of persons in their professional roles. (p.52)

This unified conception was put under strain by a number of external circumstances. The education authority adviser and some of the teachers were uncomfortable (at least initially) with the 'research' tag, and the research committee within my own institution would not have supported me with additional time had I presented the project to them as being 'only' about curriculum development or teacher professional development.

Project Activity	Duration & Timing	Purpose
Project meetings.	Half-day, monthly in the first year. Thereafter termly, half-day or full day.	To meet, plan, discuss, co-ordinate trialling of materials, review progress.
Small steering group meetings.	Half-day. Once per session.	To discuss funding, progress, dissemination, evaluation, future directions.
Informal meetings with the educational adviser.	Following most project meetings.	To provide a regular update on progress, to aid strategic planning.
Departmental meetings in schools.	Frequent (usually weekly).	Items on the agenda relating to the project would be discussed and fed back to the project.
Writing teams.	Throughout the project.	To draft and revise materials.
Project workshops.	One or two day. Once per session.	To set in train small group activities on development, assessment and evaluation.
Working groups.	Whenever the need arose.	To enable a small group of teachers to do concentrated work in one area (such as assessment).
Inservice courses in Lanark Division.	One-day. Once or twice per session.	To disseminate to other computing teachers. To extend (at certain times) an invitation to join the project.
Other short inservice courses and teacher conferences.	No set pattern, during third and fourth year of project.	To disseminate to other groups of teachers (on their invitation) and gain feedback.
Research conferences.	As above.	To disseminate to the research community and gain feedback.
Reciprocal visits by teachers.	During the second and third year of the project.	To enable teachers to observe other project teachers with programming classes.
Visits to project classes, by me.	During the first – third year.	To observe programming classes and to discuss any problems with teachers individually.
Visits to one class over a session, by me.	During the third and fourth year.	To participate in teaching and observe the materials in use over a continuous period.

Table 3: Programming project activities.

According to Elliott, the role of any 'outsider' should not be to, 'impose on practitioners a hegemony of specialist expertise,' (ibid., p.54) but rather to nurture this unity by supporting and facilitating reflective educational practice. Collaboration empowers individual teachers by enabling them to critique the curriculum structures which shape their practice and to negotiate change within the system. He comments, 'For the isolated teacher ignorance is bliss. It allows such a teacher to sleep at night by living under the illusion that the improvement of practice is largely a matter of developing technical skills.' (ibid., p.55)

Coolican (1994) notes that collaborative research is not without confrontations, but the idea is to build on these natural differences constructively. He envisages that the 'outside' researcher would play the leading role at the beginning of a project with participants gradually taking on a more central role in the progress of the research. There are clear dangers in this approach of the goals of the project being hijacked by the 'outsider' and of the intended shift in locus of control never occurring. I preferred to work with teachers 'from the inside', sharing my thoughts and ideas with them and allowing the project to develop naturally. Fullan's (1993) account of the individual teacher as an effective change agent – he or she needs to set personal goals, to become immersed in the activity, to pay attention to what is happening, and to learn to enjoy immediate experience without becoming perplexed about external circumstances – provides a better model for empowerment. He states: 'With all the emphasis on collegiality and collaboration, it is easy to neglect the necessity of learning to think for oneself.' (p.139)

This balance of 'inner learning' and 'outer learning' was reflected in our attempt to convey something of the flavour of the project at the Scottish Educational Research Association Conference in 1992 when one of the project teachers and I spoke of a supportive, collaborative environment that would enable individual teachers to experiment with their own ideas within an agreed framework (Kirkwood, 1992).

3.2.2 *Project activities, processes and evaluation*

The project extended over four school sessions and involved a wide range of activities (table 3). All teachers from the participating computing departments were invited to take part in group forums – it was suggested that a rota system should operate when it wasn't possible for more than one teacher to get release. Notes were taken by teachers (again using a rota system) of any group

discussions and these were circulated to every department. A full record of all project outputs including notes of meetings has been retained.

Not all teachers wished a full role in decision making or development and some contributed through trialling the resources and providing feedback. The resources were distributed to all computing departments in Lanark Division as each of the four study units was completed, supported by inservice and teachers' notes[11]. At the end of the first and second years of the project there was an opportunity for some other computing departments to join the project, and each 'new start' was paired with one of the existing departments to provide support. As the project grew, to involve eighteen departments in the final two years, project teachers took on a bigger burden of administration (e.g. one teacher sorted out any problems with the distribution of materials for trialling).

Evaluation was on-going, formative, and conducted within the project. Its main focus was on the effectiveness of the measures that we were implementing in the classroom, rather than directly on the effectiveness of 'teacher learning', which is something that we reflected on frequently, often in very informal settings – after project meetings etc. – but did not take any formal steps to measure. However it would be possible, if resources could be found, to assess the impact on teacher learning in the longer term.

Evaluation efforts were mostly channelled into field testing of the learning resources. To begin with, this was relatively straightforward. We put in place a first draft of the first unit and used a variety of tools (outlined below) to evaluate it. We then put in place a first draft of the second unit, as pupils were ready to move onto it, and a second draft of the first unit with the next year group, and so on. Field testing of all four units became a complex process requiring careful co-ordination, especially as it involved quite large numbers of both S3 and S4 pupils (e.g. the second draft of the first unit was trialled with nearly seven hundred S3 pupils) and the writing and revision of the materials had to go on in the background. However the advantage of doing it this way was that the lessons learned from field testing of the earlier units could be applied to the writing of the later units which therefore needed fewer redrafts.

The evaluation tools comprised:

- teacher diaries and written annotations to the materials;

---

11 A wider distribution to all education authorities in Scotland also took place through an informal network of educational advisers.

- notes from school departmental meetings;
- class records of progress rates and attainment grades;
- noting the actual ratio of pupils to computers to highlight problem cases (e.g. that in one classroom there were on several occasions only seven 'working' computers for 19 or 20 pupils);
- questionnaires (with open and closed items) completed by pupils at the end of each study unit, in which pupils appraised their own learning and a range of aspects of the learning resources and environment;
- observations of classes (but not formalized by any observation schedule) and short reports of observations and discussions with the class teacher;
- evaluation workshops to collate and analyse the data;
- regular feedback of information from evaluations to project teachers to inform discussion and decisions;
- presentation of drafts and re-drafts of materials at project meetings by authors, to obtain feedback;
- presentations and reports by teachers on methods that they had investigated;
- presentations at 'outside' forums to obtain external feedback.

The class records and pupil questionnaires are discussed in detail in chapter 4, in relation to their use in the case study setting.

The evaluation tools were therefore wide-ranging to encompass process and end-product measures, qualitative and quantitative data, cognitive and affective outcomes, and a range of perspectives to include that of all participants (my own, all teachers, all pupils). The scope of evaluations also varied from the micro-level (such as the very specific points on wording, layout, etc. that formed the annotations to the materials) to the macro-level (such as the collated data from pupil questionnaires).

There was no adoption of 'quasi-experimental designs' or summative evaluation measures, as Baron (1987) recommends (she discusses pre- and post-test comparisons and the use of a control group). It would have been extremely difficult to negotiate the mine-field of such methods where the comparability of findings and the ethics and practicalities of experimentation

are always at issue, and Elliott's concern about imposing a 'hegemony of expertise' on teachers is also pertinent. This matter is given further, detailed consideration in chapter 4 in relation to the case study design. With regard to summative evaluation, where the purpose is to determine whether, 'a programme should be continued or terminated,' (p.223) this judgement, when related to the continued use of the learning resources, would be made by each computing department in the light of changing circumstances such as upgrades to equipment and syllabus reviews. The evaluation procedures ensured that the project was under constant review, thus enabling a case to be made for continued funding annually. Therefore there was no pressing need to conduct a summative evaluation at the end of the project (and, in any case, there was no additional funding available for it).

In terms of the cyclical model of action-reflection that is envisaged by most writers on action research (see, for example, Elliott's (1991) references to Kemnis's 'spiral of cycles' in chapter 6 of *Action Research for Educational Change*), a number of such cycles, overlapping in time, operated within the project. Focused attention was given to the design, implementation and evaluation of a range of aspects of the learning environment at different stages of the project, such as on summative assessment, the needs of able pupils (see the earlier reference to the SOED project in 3.2.1), or the development of the next study unit.

Some teachers pursued their own small investigations within the project, examples of which were:

- on how to help pupils overcome difficulties with accessing their knowledge of language syntax;
- on providing customized prompts for individual pupils (one teacher used art-work to produce prompts in the form of wall posters and inserts for the pupil's folder);
- on a voluntary homework scheme, used in conjunction with a range of approaches to setting or negotiating progress targets with the class and individual pupils.

3.2.3 *Dilemmas for teacher researchers*

Elliott (1991) discusses a number of dilemmas that teachers researching into their own practice commonly face, and suggests various ways of resolving them. I shall discuss briefly those which seem most relevant to the

programming project, under the following headings: encouraging pupils to critique one's professional practice; sharing data with professional peers; negotiating the different positions and roles of teacher and researcher; and finding time to undertake research.

3.2.3.1 *Encouraging pupils to critique one's professional practice*

When a teacher encourages pupils to critique his or her professional practice, does this undermine the teacher's professional status, and by implication, that of other teachers in the school? Might pupils volunteer criticisms of other teachers who are unprepared to receive them? The dilemma here arises from the conflict between the value of critical openness to pupils and respect for the professional expertise of colleagues. This is sometimes resolved by setting strict limits on what pupils can comment about: 'Do not mention other teachers by name.' 'Do not talk about specific lessons, including mine.' etc. (ibid., p.59) Informing colleagues of the professional purposes served by such critiques provides an alternative way out of the dilemma.

The items in the pupil questionnaires were very specific to programming lessons and were legitimized by the intention to bring about improvements in learning and teaching processes. The initial concern among project teachers was pupils' willingness to take the exercise seriously and to respond thoughtfully – it was not a situation that pupils or teachers were accustomed to and perhaps some pupils would use it as an opportunity to show disrespect? However pupils' actual comments were found to be, almost invariably, constructive, and thus the pupil data came to be highly valued within the project. The advantages of pupils reviewing their learning were also recognized by teachers (Kirkwood, 1996, p.29):

> It makes the pupils think about what they've learned and how well they've understood the work covered in the booklet, and it also gives an indication of the remediation required. (project teacher)

3.2.3.2 *Sharing data with professional peers*

When sharing data with professional peers ...

> Problematic areas of practice become exposed, and the practitioners operating in them become vulnerable to punitive attitudes expressed by self-styled experts who promote this image of themselves by pointing the finger at others. (Elliott, 1991, p.61)

There were a few such 'self-styled experts' who opted to join the project at the beginning of its second year. It was hoped that the spirit of critical openness

surrounding the group's activities would be 'caught', not 'taught'. Through stating the rules of discourse explicitly the new teachers were successfully integrated into the project.

3.2.3.3 *Negotiating the different positions and roles of teacher and researcher*

The problems of negotiating the different positions and roles of teacher and researcher are evidenced in the dilemmas surrounding the use of data. Teachers have constant access to 'insider' information, and how are they to deal with such information in their roles as researchers? Elliott writes of the need to develop procedures and strategies for protecting individuals from possible misrepresentations and misuses of sensitive data, such as cross-checking of accounts, giving individuals the opportunity to reply to accounts of their activities, presenting alternative descriptions, interpretations and explanations of events, and consulting individuals about the contexts in which their actions and views are represented and reported. These measures are in the interests of fairness, accuracy and comprehensiveness, and are important to establish trust in the *researcher as insider* (ibid., p.64).

These concerns are most to the fore when the methods of investigation place a spotlight on particular pupils. One of the matters that we deliberated about was whether the pupil questionnaires should be filled in anonymously, or whether the pupil's name and/or class and/or teacher's name and/or school should be asked for? The formative use of questionnaire data is hampered when the data is gathered anonymously, since the teacher only gains a general picture of how his or her class is responding to the learning environment rather than a view of how individuals are responding. We therefore adopted a reference system that enabled each pupil to be identified by the class teacher but not by anyone else, and the data for each class to be aggregated separately if required.

3.2.3.4 *Finding time to undertake research*

Elliott argues that one way that time can be found for teachers to do research is not to divorce the data gathering processes from the natural ways of gathering and processing data that teachers use routinely. Therefore making use of, for example, pupil progress records, the outcomes of summative assessments and notes from departmental meetings, provides an efficient means to accrue data. Some of the other data gathering tools (diaries, annotations to materials etc.) were designed to enable teachers to make brief, timely and accurate recordings, and to introduce to them a range of evaluation approaches that could be adopted whenever any new teaching methods or materials were being

introduced. None of the data gathering methods was considered by teachers to be intrusive (Kirkwood, 1996).

3.3 THE LEARNING RESOURCES

The learning resources (Kirkwood *et al.* , 1992a; 1993; 1994) are designed to support resource-based teaching rather than teacher-led instruction, and also to support differentiated teaching since they are intended to cover foundation through to credit level content and to be used by pupils across a range of abilities.

The components of the resources are designed to serve a variety of learning, assessment and evaluation purposes and split into pupil materials and teacher materials. The pupil materials incorporate those features that have been identified in chapter 2 as being important to develop understanding, problem solving, higher order thinking and metacognitive skills. The teacher materials provide guidance on the management of the resources, pedagogical goals, and coursework assessment.

The teaching method is problem-based, and therefore selecting problems to build a coherent programme, and developing the methods of instruction in programming and problem solving, formed a major part of our deliberations when constructing it.

3.3.1 Resource-based teaching

The reasons for adopting resource-based teaching were as follows:

- To enable learning to be self-paced. The appropriate pacing of whole class lessons is very difficult in mixed-ability or broad-banded (foundation/general or general/credit) settings which are the most common class organizations within the subject.

- To provide time for each pupil to gain competence, understanding and confidence in all aspects of their learning (this is a related point to the first one). Each newly introduced aspect is likely to feature in later solutions, and therefore there is no benefit to be gained from accelerating some pupils beyond a pace that they can cope with.

- Problem solving cannot be done 'to order' – it involves puzzling and sometimes going down blind alleys. Therefore pupils' problem solving attempts should not be tightly orchestrated by the teacher (e.g. within a whole class lesson the teacher may plan to discuss a solution at a

particular point in the lesson irrespective of whether each pupil has arrived at a satisfactory solution independently).

- Resource-based teaching accords pupils much more independence from the teacher – the pupil does not have to wait for the teacher's instructions or explanations to work on tasks – and so there is the opportunity to develop strategies for self-regulated learning.

However not all resource-based programmes appear to work successfully. Three common criticisms of resource-based teaching are that the management of the materials consumes the teachers' time, monitoring of pupils' progress is complex, and pupils tend to work in isolation from each other and the teacher (HM Inspectors of Schools, 1996; Simpson and Ure, 1993).

These problems are mainly located in the design of the particular resources and the learning environment. Many schemes are over-elaborate and therefore difficult for both the pupil and teacher to manage, collaboration may not have been planned for so that pupils lose the benefits of learning from each other, and teachers may attempt to do all of the checking of pupils' work during the lesson, which is neither desirable nor practicable (Kirkwood, 1997b). Drever (1988) highlights the need for teachers to be very disciplined and economical in the management of their time, making contact with pupils when they most need it and giving priority to the aspects which the pupils or materials cannot deal with. He also identifies a number of other issues which need to be tackled if resource-based teaching is to be successfully implemented:

- Time appears to be critical. Attempts to speed up to fit the available time can undermine the whole approach by preventing proper self-pacing, cutting off learning before mastery has been achieved, and causing teachers to restrict their resource-based teaching to a minimum.

- The design of materials should be to engender fruitful interactions between pupils, resources and the teacher, rather than to enable pupils to work unaided.

- Research evidence shows that pupils may simply 'go through the motions' of one activity after another.

- A strength of traditional teaching is that teachers can lend coherence to a topic. This can be lost, and with it, retention ('they do it, check it,

forget it') if resource-based materials are designed on an instructional objectives model.

The last three issues concern the design of the resources – primarily the nature of the activities encompassed and the coherence of the teaching programme. Within the programming project these issues were taken on board at the planning stage. The time factor was the most difficult to overcome. Resource-based programmes typically take about 20 per cent longer than those taught by traditional methods (Drever, 1988), and this is borne out by our own data where the average class time spent on programming within the SOED project was in excess of 52 hours, which is approximately 30% above the recommended time of 40 hours.

Lack of time was addressed through a range of measures, which were applied differentially by project teachers according to the circumstances and their preferred teaching styles. These included:

- careful monitoring of individual progress rates and (some teachers) negotiating individual targets with pupils;
- keeping a careful watch to identify pupils struggling with the work and providing any necessary assistance (since lack of comprehension or chaotic working habits are likely to slow progress considerably);
- (most teachers) encouraging pupils to do extra programming work in their own time – at lunch times and after school with supervision from the teacher, and at home (written work);
- not attempting to provide evidence of pupil achievement across all aspects of the programming content at foundation, general or credit level (but meeting the SEB's requirement for broad coverage);
- (most teachers) allocating more time to programming and reducing the time spent on other topics; and
- (some teachers) advising certain pupils to by-pass those problems that were not needed for consolidation, thus addressing the criticism in the report by HM Inspectors of Schools (1993) of approaches which force all pupils to work through a common set of activities whether they need to or not (see 1.2.3).

The majority of these measures are desirable even in circumstances where there is less pressure for coverage. The report of the SOED project on

accelerating progress in learning (Kirkwood, 1996) documented our attempts, which were not entirely successful, to grapple with this issue, a copy of which was sent to the SEB computing studies panel.

There has been little training for teachers on the design, appraisal and use of resource-based materials (Drever, 1988). The programming project provided this opportunity for participants and indirectly for other teachers who were using the learning resources with classes and following the guidance within the teachers' notes.

3.3.2 *Differentiated teaching*

The traditional view of school learning 'explains' variations in attainments between pupils as a natural consequence of pupils having different amounts of general ability, and general ability is perceived as something intrinsic to the pupil and beyond our influence (Drever, 1985). Bloom has dismissed this explanation as 'no explanation'. Drever explains Bloom's ideas as follows:

> But how do we know that they differ in ability? As a rule we infer this from the differences in attainment. The 'explanation' is no explanation. But what it does is important: it allows us to take for granted the quality of our *instruction*. (ibid., p.60)

Bloom (1976) substitutes a different model in which the idea of general ability is abandoned in favour of specific abilities. He argues that a pupil's chance of success on a learning task will depend on three things: whether the pupil has already learned the specific skills and knowledge that the task requires; whether the pupil has an interest in learning from the task (which is not to be interpreted narrowly as things the pupil is interested in); and on various aspects of instruction. The attainments that result from the task add to the pupil's stock of specific abilities and raise the chances of future successes.

The important point about Bloom's model is that it deals with variables that can be altered rather than with a fixed quantity (general ability). He argues that if we can identify the alterable variables that can make a difference to children's learning, this will do much to explain the learning process and even more to directly improve the teaching and learning processes in schools. Bloom's view is therefore an optimistic one.

Drever has investigated Bloom's model with S1/2 pupils in relation to mastery learning. In spite of teachers' well grounded concerns about Bloom's 'utopian vision' (for example, that it does not give adequate recognition to the affective

dimension of learning and overplays the formal, cognitive curriculum), he concluded there was no reason to doubt Bloom's central premise. Also whenever teachers could be induced to suspend belief in 'general ability' and act instead on Bloom's hypothesis, they began to look critically at curriculum and assessment and to experiment creatively in their classrooms, and they reported an improvement in pupils' motivation and performance.

However this account (below) by a project teacher of the attitudes of his pupils to their programming work, demonstrates that it is not always easy to bring about such a suspension of belief:

> One or two appear to become bored if they spend a number of consecutive weeks programming – though more intelligent tend to improve in attitude and motivation as the problems become more challenging.
> (project teacher)

The use of the labels 'credit level', 'general level' and 'foundation level' attached to pupils and classes, rather than solely to different award levels in an examination, does little to help this problem.

Through using differentiated approaches teachers seek to address the individual learning needs of pupils. According to Simpson and Ure (1993) the way in which teachers think about differentiation and go about doing it is influenced by their beliefs and assumptions about ability or intelligence. When Simpson and Ure investigated differentiation in Scottish secondary schools they identified two models of differentiation operating. One of these models (labelled by the researchers as 'measure and match') has as an assumption that a stable, underlying characteristic of pupils is the key determinant of pupils' competencies (for example, intelligence or mathematical ability), and that this characteristic can be reliably measured at a point in time. A further assumption is that a match can be made between the competency of the pupil and the level of difficulty of the curricular materials or course, and that this match can subsequently be fine-tuned by summative assessment. This model was found not to deliver good differentiation, according to pupils' perceptions. The second model ('pick and mix') works on different assumptions. It assumes that the competencies of pupils are determined by a complex range of factors (motivation, classroom relationships, past learning experiences etc.), and that these competencies will continue to be influenced by these factors, and therefore measures at any one point in time are useful, but not critical (they are of transient interest). Furthermore it assumes that a range of learning needs has to be taken into account when allocating work or responding to learning outcomes, and that differentiated materials should cater for this range and be

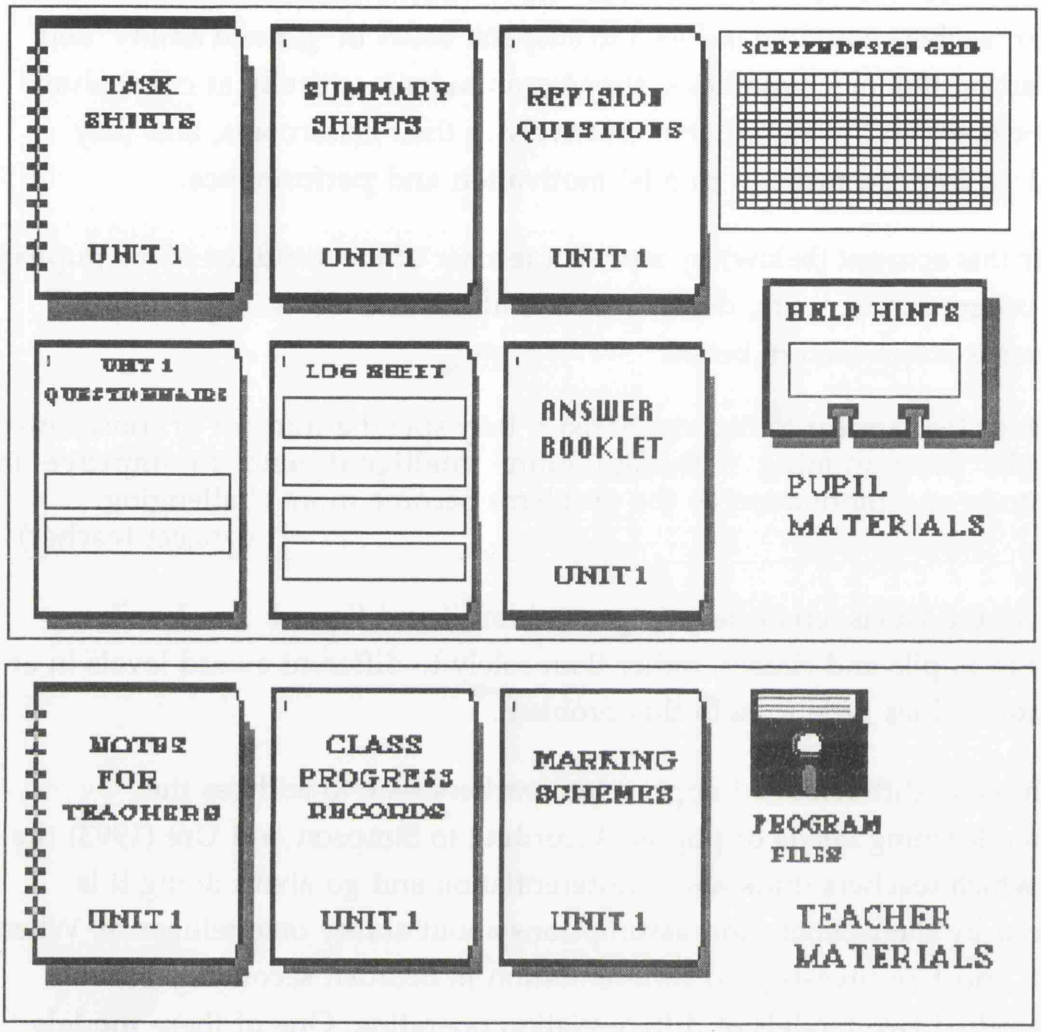


Figure 1: The learning resources.

accessed as and when appropriate for individuals or groups. Not surprisingly, this model was found to be far more successful by pupils.

Bloom's model underpins the approach to differentiation adopted within the programming course. We would seek to foster in teachers (and pupils – see 2.6.1) an incremental view of intelligence. Our approach is closer to Simpson and Ure's 'pick and mix' model than to their 'measure and match' model. The content, pace and support for learning can be adjusted to suit the needs of the individual learner, but none of this adjustment is expected to be delivered through the learning resources alone, with the teacher acting in some distanced role to oversee events, dish out worksheets and act as an assessor. The teacher's role is, first and foremost, pedagogical.

The differentiation model can be summarized as follows (to encompass the earlier points from 3.3.1):

- learning is self-paced;
- there is flexible provision for acceleration within a common programme of activities which engage pupils in problem solving and enquiry;
- pupils are encouraged to interact and to initiate collaborations with each other, thus ensuring that the social context is exploited fully to maximize learning opportunities;
- the teachers' role is mainly pedagogical, to teach, model and guide pupils' solution attempts through targeted interventions with individuals and small groups;
- there is on-going formative assessment of pupil performance and self-appraisals by pupils of their progress, which includes aspects such as understanding and confidence;
- individual pupils share decisions with the teacher about their next steps in learning.

3.3.3 The components of the learning resources

The components of the learning resources are shown opposite (figure 1). There are four study units.

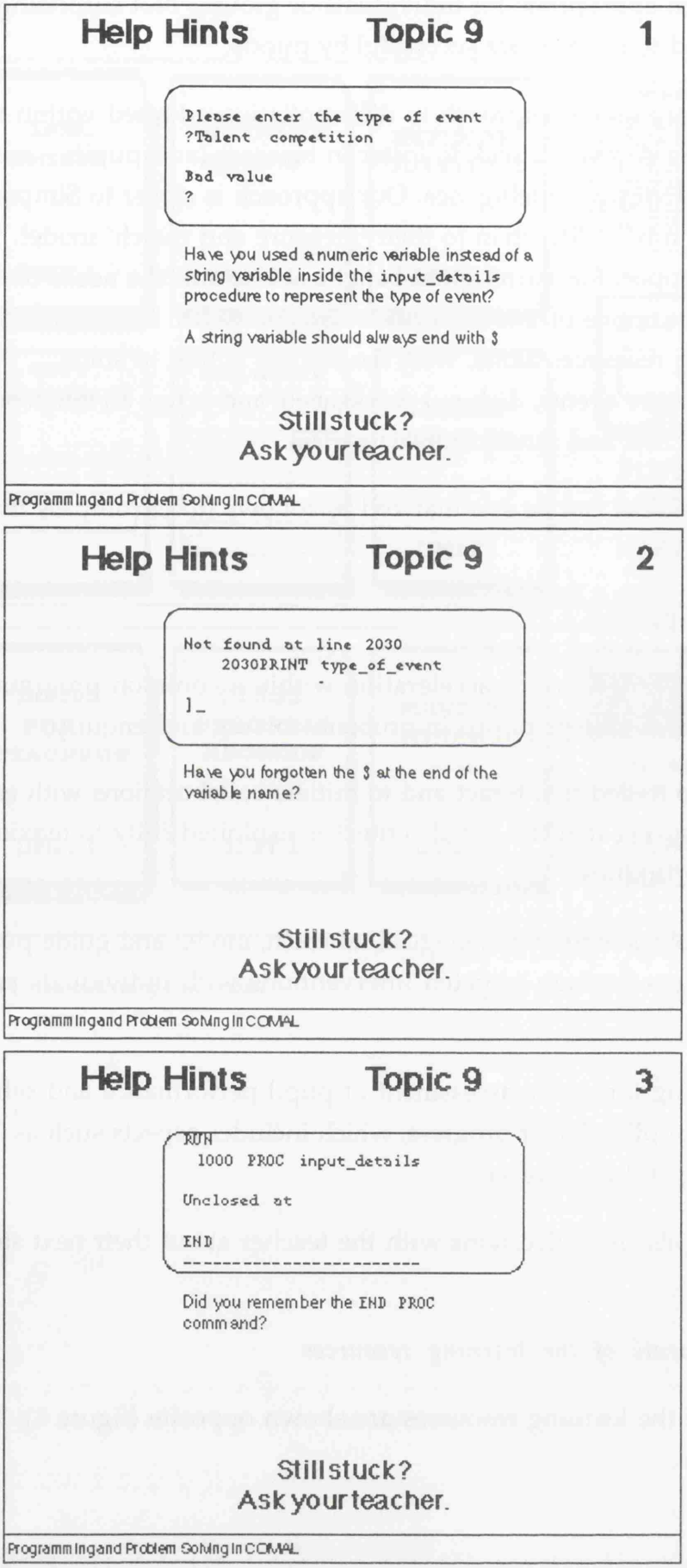


Figure 2: Three 'Help hints' from topic 9.

3.3.3.1 Pupil materials

Each **task sheet** booklet contains four or five topics. These provide the instruction in programming and problem solving. Summative assessment tasks are fully integrated into the units (e.g. there is one in topic 3 and another in topic 5 of the first unit). At the end of each topic the pupil is directed to the teacher to have his or her work checked. In units 3 and 4 each topic ends with a set of ideas for pupils to explore which can involve modifying a solution, experimenting with new techniques, applying ideas to related problems, and generating and/or contrasting alternative solution methods (including using other software tools such as a spreadsheet or database).

The **design grid** is a tool for doing an accurate screen layout (it marks out the row and column positions).

The **help hints** provide a diagnostic aid for debugging programs. They provide hints, not answers to problems, and are designed to enable pupils to match the circumstances of an error to a particular 'hint' (figure 2). They are available for topics in the first two units, when pupils are least experienced at recognizing and dealing with errors.

The **revision questions** and **summary sheets** are designed to be used together. The revision questions provide an additional set of problems, similar to those in the task sheets, for reinforcement or revision. There is no requirement to test any of the programs at the computer which enables pupils to work on the revision questions at home. The summary sheets are for reference and revision. They summarize and exemplify the key teaching points within each topic.

The **answer booklet** provides a neat format for pupils to record their answers to the task sheets and their progress from day-to-day. It is retained in a **folder of work**, together with any printouts, design grids and answers to the revision questions.

The **log sheet** provides a format for pupils to record answers to each summative assessment task. These are retained in an **assessment portfolio** which contains a rolling sample of evidence for each pupil (only the two most recent, successful items are picked as evidence towards the programming coursework grade).

Class 3C	10/9/95	11/9/95	14/9//95	...
Andrew Burns		1	x	
Susan Davies	1		2	
Joan McLean	1	2		
...				
Key: x – Absence. 1 (2 ...) – Topic 1 (2 ...) completed.				

Table 4: Class progress record.

The **questionnaire** prompts pupils to review their learning over each unit and provides feedback to the teacher on pupils' reactions to a range of aspects of the learning resources and environment.

The pupil resources therefore support self-directed learning through enabling pupils to: work on problem solving tasks, either independently or collaboratively with other pupils; gain regular feedback on their performance from the teacher; access reference material; monitor their own progress from day-to-day and over a longer timescale; build a detailed and organized record of their work; consolidate, revise and review their learning; and measure their attainment against syllabus targets at regular intervals.

3.3.3.2 Teacher materials

The main aspects covered by the **notes for teachers** on each unit are as follows: information on the programming project; an overview of the content and teaching methods; an explanation (as above) of each component of the resources with instructions on duplication; syllabus coverage; guidance on – preparation, classroom management, how and when to assist pupils, monitoring progress, coursework assessments, and individual topics (learning outcomes, estimated time to complete etc.).

The **class progress record** (table 4) is used to record, for each pupil, the date of any absences and of completion of each topic.

A **summary class record** indicates the number of periods to complete each topic.

Marking schemes conform to the SEB requirements, and set out the criteria to be met in each solution.

Program files enable pupils to load some programs or parts of programs.

3.3.4 Selecting problems to include in the course

The task sheets provide the instruction on programming and problem solving, organized into discrete units and topics which pupils tackle in sequence. The units and topics are not designed to take an equal amount of time to do.

Each topic is centred around a problem or sequence of related problems. The topics introduce systematically a range of programming concepts and principles, standard algorithms, problem solving processes, COMAL language features and process skills (see appendices 1a and 1b). A topic places an in-depth

focus on a problem or class of problems which lends it internal coherence. Having developed or examined a solution, pupils may be asked to document it, evaluate it, suggest improvements to it, amend it or enhance it in some way, generate an alternative solution, contrast alternative solutions, tailor the solution or re-use elements of it to solve a new problem, or review the problem. Thus higher order thinking is embedded in all aspects of the teaching within a topic.

For such a problem-based approach to work successfully, it is essential to find 'good' problems to form the basis of the teaching in each unit. Such problems came to be viewed within the project as satisfying all of the following criteria:

- problems that provide powerful demonstrations of techniques;
- problems with mileage (or generative potential), that can be introduced, developed and extended naturally;
- problems that will engage and sustain pupils' interest;
- problems which provide scope for pupils to build their own ideas into the solution;
- problems that are well matched to pupils' stage of learning and experience – each problem should be challenging, but not so complex that it is likely to lead to frustration and boredom;
- problems that are worthy of serious treatment, to justify the time and effort that pupils will expend, and to ensure that a genuine sense of achievement is experienced when a solution is found.

The types of problems to be avoided are in plentiful supply in many existing teaching programmes, such as the task of devising a weekly rota of duties for the Seven Dwarfs, chosen by the SEB (1993b) as an example of a suitable coursework assessment task at credit level. That this is not a suitable problem is illustrated by the question: how is the solution to be designed and evaluated in terms of user requirements? It is merely a rather difficult exercise in coding which constitutes a missed opportunity for pupils to do some worthwhile programming.

Pupils' limited experience of programming places constraints on the range and complexity of problems that can be presented in the earlier topics, and, in particular, it is difficult to find authentic contexts. However by providing scope for pupils to build their own ideas into the solution it is possible to engage

pupils in 'real' programming from the outset. This can happen when pupils select a context (e.g. what their poster or quiz or survey will be about), gather or choose data to enter to their program, design output to the screen (e.g. what their ship or robot design will look like), design user-documentation and test their program on a real user, think out a solution without prompting or assistance, or evaluate their solution and suggest enhancements or modifications.

Pupils already possess a well-developed understanding of algorithms, through their everyday experiences – many children's games and sports are governed by algorithms – and, more formally, through mathematics. One can select varied contexts that build naturally on pupils' existing knowledge (such as in topics 8, 14 and 15) to provide bridges to the new domain of programming (transfer in), and to enable pupils to experience at first hand the power and versatility of a programming language as a tool for solving problems.

The problems were chosen, and the topics developed by the authoring team of each unit, using a staged process of: brain-storming to identify candidate problems, applying the selection criteria to narrow down the choice, each author producing a first draft of one or two topics, gaining feedback from the other members of the authoring team and revising the topic if necessary, and trialling. The questionnaire responses were helpful to determine the successful and less successful problems during trialling, and some topics were ditched and redesigned as a consequence. I participated in each of the writing teams as an author, and undertook the task of editing the materials and producing the full set of resources.

The teaching for understanding framework of *Project Zero* (see 2.3.2) is reflected in the design of individual topics. The short narrative introduction to each topic identifies a few understanding goals and links them to previous understanding goals. The majority of performances are understanding performances since they require that pupils go beyond the givens. On-going assessment features strongly in our framework through formative assessment of pupils' performances in each topic, self-checks (pupils using the revision questions and summary sheets together at the end of a topic), and formative feedback on items included in the assessment portfolio.

3.3.5 The methods of instruction on programming and problem solving

The organization and presentation of the instruction within the pupil resources has certain important characteristics, each of which is outlined below. The later chapters of the thesis provide more detailed discussion on

these aspects, and examples of the class teacher's pedagogical interactions with pupils.

3.3.5.1 *Coherence*

The coherence of the programme arises in part from the progression within each of the aspects of the content (looking down each column of appendix 1a), for example, taking standard algorithms, topics 1–5 focus on output only, topics 6–8 introduce input – process – output and topics 9–18 introduce a range of standard algorithms which involve the use of control structures. The consistent manner in which problems are introduced and developed (see appendix 1b), and the revisiting of some earlier problems and solutions (for example, topics 13 and 18) also lend coherence to the programme.

3.3.5.2 *Fostering a careful, analytical, planned and systematic approach*

The programme is designed to foster in pupils a careful, analytical, planned, and systematic approach to the solution process. It is not intended that pupils will learn through unguided discovery since it is important for them to learn how to manage the complexity of problems and situations. In this way they will avoid some of the pitfalls that prevent inexperienced programmers and problem solvers from making progress and which often lead to frustration and giving up. A mainly top-down approach to designing solutions is adopted in the materials, and a range of testing and debugging skills is introduced to provide the skill base.

3.3.5.3 *Problem solving does not have to be postponed until after the 'basics' have been covered*

There is no postponement of problem solving until after the 'basics' of the programming language have been covered. Teaching on problem solving processes is first introduced through very straightforward problems, and pupils' skills are developed and honed through exposure to a range of problems and solution strategies.

3.3.5.4 *Pupils build a store of useful and accessible knowledge of previously solved problems*

The course is not organized in the conventional way around language features or details of syntax, rather it is designed to enable pupils to recognize the similarities and differences amongst problems and solution methods, and to recognize and re-use standard algorithms and elements of previous solutions. Pupils are building a store of well-integrated knowledge on previously solved problems and solution methods which they should be able to access in future.

3.3.5.5 Pupils experience a range of methods of synthesizing solutions

A range of methods of synthesizing solutions is introduced – merging (topic 9), tailoring an existing solution (topic 10), nesting (topic 12), and abutment (topic 13). The distance between problems is increased gradually as pupils' ability to recognize common programming goals and to synthesize algorithms (or plans) develops.

3.3.5.6 General principles of programming knowledge and procedures are stressed

Language features and programming concepts and techniques are introduced gradually and in a manner which stresses the general principles of their use, in order to build programming competency and to foster near transfer to other programming or software applications environments. (The relationship of the content of each unit to the general programming specification is set out in appendix 1c.)

3.3.5.7 There is explicit teaching on problem solving processes

Problem solving processes are taught explicitly and rehearsed frequently. These are introduced within the framework suggested by Polya (understand the problem, make a plan, carry out the plan, look back) but using the more formal language of the Standard Grade practical abilities element (analysis, design, implementation, evaluation). This explicit treatment is designed to develop in pupils a common language of problem solving and a heightened awareness of problem solving processes. Pupils' capacity to solve programming problems should be enhanced, as should their ability to transfer problem solving skills to non-programming contexts.

3.3.5.8 The solution process is modelled

In most topics the solution process is modelled in writing. The solution unfolds in stages (the breakdown of the 'posters' problem in 5.3.3 illustrates the approach) with accompanying discussion of the concepts and methods employed. Pupils contribute to the solution by applying their previous knowledge and using the new information given in the topic. This provides a mechanism for the learning resources (and also the teacher, taking his or her cue from the resources, since the written discussion must be contained to an amount that the majority of pupils will be prepared to read) to demonstrate the flexible application of strategies to a variety of programming contexts. The teacher's modelling of problem solving processes may take the form of *coaching* of individual pupils or small groups.

Implicit in modelling is that the pupil should gradually take over control of his or her own learning. This is encouraged through providing opportunities for further practice and feedback on similar problems, the gradual withdrawal of support (written guidance, hints or partial solutions), and the teacher encouraging pupils to self-prompt (see 4.5.5).

3.3.5.9 A wide range of thinking and metacognitive skills and strategies is encompassed

These include the skills and strategies which arise across a range of programming contexts: analysis, planning, program design, testing and debugging, evaluation (assessing the adequacy of solutions, generalizing solutions, comparing and contrasting solutions, considering alternatives, suggesting improvements), reviewing learning, monitoring and checking, making and testing predictions, making logical deductions, drawing inferences, generating ideas, and viewing a solution from different perspectives (that of the user and programmer).

Some strategies require a considerable degree of judgement and skill to apply. For example, applying a top-down design process requires: the ability to perceive the whole problem, to break the problem down in a manner which makes it amenable to solution using the available tools of the programming language, to recognize the main component parts of a solution, to recognize which of these parts needs to be refined, to develop the refinements in parallel (avoiding refining some parts to a very detailed level while others are left unrefined, since this can result in going down blind alleys), to represent the solution clearly and unambiguously at all of its stages of development, and to check that the solution is accurate at each stage. There is a whole range of related skills that need to be orchestrated by the learner to apply a complex strategy such as this, which points to the need for each of these skills to be introduced separately and then to be combined in use across a range of contexts, as Schoenfeld (1985) recommends in relation to mathematical problem solving.

There is a spotlight on particular skills and strategies at various points in the findings, for example, on testing and debugging (in 5.3.7 and 8.5), monitoring and checking (in 5.4.2), viewing a solution from different perspectives (in 7.2.8), planning strategies (in 8.4), and contrasting alternative solutions (in 8.6). Chapter 6 examines their integrated use in the context of coursework assessment tasks.

3.3.5.10 *Relationship of the teaching method to cognitive apprenticeship models of learning and teaching*

Models of learning and teaching based upon *cognitive apprenticeship*(see 2.6.3) identify the processes of modelling, coaching, scaffolding (propping up the learner's early solution attempts), fading (gradual withdrawal of this support), articulation, reflection and exploration as component processes which combine and interact to facilitate learning. The methods of instruction within the resources incorporate some of these features (modelling, scaffolding, fading), and the presence of the others is dependent upon the teacher adopting an appropriate pedagogical role.

Other important features of cognitive apprenticeship are its use of the social setting to foster the development of independent thinking skills and the setting of authentic tasks which increase gradually in complexity. These features are also present in the design of the learning environment and are discussed more fully in later chapters of the thesis.

4 CASE STUDY DESIGN, CONDUCT AND EVALUATION

4.1 CHAPTER OVERVIEW

This chapter describes the case study design, its conduct and evaluation. The case study presents a combined learning and research environment in which the progress of one Standard Grade (S3/4) class is studied closely over an extended period from September 1993 to March 1995 while they are learning to program. The pupils were aged thirteen or fourteen at the beginning of the study.

The criteria for selecting the case study class are discussed and the case study setting is described. The procedures for gaining permission to conduct the research and the manner of gaining consent from participants are outlined. The ethical and practical considerations surrounding the participation of school children in the research are discussed throughout the chapter.

Key facets of the learning environment are presented viz., the scheduling of programming classes, advice given to pupils on conducting their work, the manner of utilizing the learning materials within the study, the creation of an appropriate climate for learning, and the framework for pedagogical interactions.

The different methods of data collection are discussed, all of which serve an educational purpose in addition to their role as research tools. The criteria which are used to evaluate the overall success of the learning environment in the case study class are enumerated, and these are linked to the different sources of evidence. As well as focusing on the whole class the evidence will focus on individual pupils, and through this I will gain an understanding of how the learning environment has impacted on pupils' thinking and learning processes and how these have evolved over time. There will be a fuller description of the methods of evaluating the case study evidence in the following four chapters in which the findings are presented.

The review and discussion at the end of the chapter examines the strengths and weaknesses of the case study design in comparison to other research methods.

4.2 CRITERIA FOR SELECTING THE CASE STUDY CLASS

The case study class was selected from the S3 classes that were coming 'on stream' in August 1993 as part of the continuing process of field testing and

evaluating the learning materials within the programming project. The selection of the class was based on a number of criteria relating to the school, the qualities and experience of the class teacher, and the composition of the class. The criteria are outlined below. In the next section details of the case study setting are presented.

There is an element of subjectivity in both the selection of the criteria and in their application. The criteria do not offer a general prescription to be applied by other educational researchers when selecting a case as a focus for their research, rather they reflect my judgement about the factors that were important in the light of the research purposes and the unique circumstances surrounding the study. As Kelly (1989) has observed, research purposes are neither value-free nor neutral, and the particular values that they reflect may not be shared by other researchers.

4.2.1 Criteria relating to the school

The main factors that I considered when selecting the school for the case study setting were: (a) whether the research aims were likely to accord well with the school's priorities; and (b) the quality of the contribution of the computing department to the work of the programming project (the level of computing resources was not a key concern since the case study would make no additional demands on resources, and it was assumed that the department would be adequately resourced to offer the Standard Grade course).

4.2.2 Criteria relating to the qualities and experience of the class teacher

I considered that it was important for the class teacher to be familiar with the learning materials and confident about using them with pupils. He or she should also support the aims of the research. These factors are important to address issues of *timing*which affect the evaluation of curricular innovations. Harlen (1994) observes that because timing factors are often ignored in evaluation studies short-term findings can appear to be negative. She argues that teachers need time to familiarize themselves with new materials and to implement them in a manner that is consistent with the intentions behind their development, and it also takes time for teachers to develop self-confidence when considerable changes are required of them in their practice. The duration of the study also allows time for pupils to adapt to new learning approaches and to gain self-confidence.

Given the long duration of the study, a successful research partnership will require that the class teacher and I are comfortable with each other's presence

in the classroom. Insights must be shared as part of the process of on-going evaluation of the progress of the research. Research interventions must also be jointly planned to ensure their smooth running and to minimize any disruptions to pupils' education. Hence there is a requirement for the class teacher to adopt an active role in the research, which in turn requires a willingness to innovate, a critical disposition and a rigorous approach to evaluation.

4.2.3 Criteria relating to the composition of the class

The main aspects that I considered relating to the composition of the class were its gender balance and the range of pupil abilities within it. A reasonable gender balance is necessary to ensure that gender differences are not ignored in the research and to avoid bias towards one gender. It is important to include a range of pupil abilities in accordance with the principle of the 'thinking curriculum' as a curriculum which extends to all pupils and not just to an academically able elite (Resnick, 1987).

4.3 DESCRIPTION OF THE CASE STUDY SETTING

4.3.1 The school and the computing department

The school from which the case study class was selected is a six year, denominational, comprehensive secondary school with a large roll of over one thousand pupils. It is situated in an industrial town within an area where there is high unemployment as a result of the decline of heavy industries; most recently, the steel industry. There are three areas designated as *Areas of Priority Treatment* within the school's catchment. This is reflected in the statistics on: the percentage of pupils who are entitled to school clothing grants in the year group (42%), which is a high proportion; and the percentage of pupils from families classified as social class 1/2 (24%), which is a low proportion (Lanark Division Quality Assurance Unit, 1995). The academic performance of S4 pupils for the two year groups that preceded the case study has been very close to the national average, according to statistics issued by the Scottish Office; in 1992, 75% of S4 pupils gained 3+ Standard grades at 1–4 vs. 77% nationally , and the corresponding figures for 1993 were 79% vs. 79% nationally pre-appeal (Scottish Office, 1993). The school has recently mounted an after-school programme of supported study in an effort to raise the attainments of pupils.

Two of the three teachers in the computing department had contributed very effectively to the programming project since its inception. I had worked in

partnership with both teachers during session 1992–3 to field test draft versions of units 2–4 with an S3 and S4 class, and I was therefore already familiar with the department and the school. The headteacher was also willing to permit the case study to be located in the department.

4.3.2 *The class teacher*

The teacher of the case study class is the head of the department. She was able to arrange her timetable to make time available after most programming lessons to discuss with me the progress of the research and to assist in the planning of activities relating to it. Her qualities were well recognized by colleagues in the programming project, in particular, her ability to generate ideas and willingness to share them and offer support to others (reflected in the notes of project meetings, from 12/11/90 onwards).

4.3.3 *The case study class*

The case study class consists of twelve girls and eight boys, each of whom had opted to study Standard Grade computing studies at the end of S2. In the computing department pupils are allocated to S3 classes depending on when they are scheduled to come (this depends on their other subject choices), and on the basis of their attainment in mathematics during S2. This class was one of three classes scheduled to come in the same block, and most of the pupils in the class were expected to gain credit or general awards in computing studies (based on the past performances of previous classes). On the basis of national performance in recent years (SEB, 1996), approximately 70% of pupils overall can expect to gain a credit or general award in computing studies – statistics for the 1995 examination indicate that 34.0% of S4 candidates were awarded a grade 1 or 2 (credit level), and 37.3% a grade 3 or 4 (general level). Hence there was likely to be quite a wide range of abilities in the case study class.

Pupils had been given a very brief introduction to programming in COMAL as part of a computer appreciation course in S2, based upon the first two topics of unit 1. Four of the pupils also had some limited experience of programming in BASIC, either through entering short programs from manuals supplied with their home computers, or from programming simple games in primary school.

4.3.4 *Resources and classroom layout*

The hardware resources consisted of eleven networked BBC computers and one Archimedes computer, together with a file server and printer server, arranged around the periphery of the classroom. There was seating in the

centre with individual pupil desks grouped in twos or threes, and a resource area at the back. The software utilized for programming was Acornsoft COMAL.

4.4 GAINING CONSENT TO CONDUCT THE RESEARCH

Written permission to conduct the research was obtained from the education authority, in accordance with established procedures[12]. Before this was granted, I had to obtain the consent of the headteacher, and the parents/guardians of the children for their sons/daughters to participate in the research. The education authority, headteacher and parents were each informed in writing about the nature and purposes of the study, and they were given an assurance that the identity of the school and of the pupils (other than their first names) would not be revealed when the research was published. Pupils' individual consent was gained prior to their participation in activities relating to the research which went beyond their normal classwork, for example, interviews and questionnaire completion, and in each case the nature and purpose of the activity was explained to them.

4.5 DESCRIPTION OF THE LEARNING ENVIRONMENT

4.5.1 The scheduling of programming sessions

The class spent either one, two or three 52 minute periods each week on programming from September 1993 to mid-October 1994, depending on the programme of activities that their class teacher had planned, and thereafter individual pupils could opt to continue with their programming, or with other practical coursework, for 30 minutes each week in class and for a further hour one evening after school (by participating in the school's supported study scheme) until the end of March. Altogether the class spent the equivalent of 81 periods, or 70 hours, on programming from September 1993 to mid-October 1994.

Pupils worked through the programming units at their own pace, and latterly to targets that they had agreed individually with the class teacher. It had been intended that pupils would spend about 55 hours on programming and that they would complete the work by June of S3, however more time was allocated in S4 to enable most pupils to reach their agreed targets.

---

12 Strathclyde Education (1993), Application and Standard Form Contract to Undertake Research in Education.

In terms of gaining other practical competences, pupils learned word processing and graphics skills during S3, and how to use a database and spreadsheet during S4. Pupils would be required to use their word processing and graphics skills to produce user documentation for a unit 3 program (rainfall statistics in topic 13). They would be required to draw extensively on their knowledge of programming concepts and principles and their problem solving skills in order to solve problems using databases and spreadsheets.

4.5.2 *Advice to pupils on how to conduct their work*

Before the class embarked on the first unit, the class teacher went over the notes for pupils at the start of the task sheets since she wished pupils to follow the advice within them. The specific points were as follows: pupils should work at their own pace; ask the teacher to check their work at the end of a topic; read the task sheets carefully and seek help if necessary; record their answers in the answer booklet; maintain their progress records; and, if there was a lot of reading or writing to do, work away from the computers at their desks. This latter point was important to ensure that on-line time was used efficiently since pupils would come to the computer with a program ready to be typed in, and it also freed up the computers.

Pupils were encouraged to collaborate freely on most aspects of the work, when analysing problems and designing and evaluating solutions (except when doing coursework assessments, which have to be completed individually to satisfy SEB requirements). They could also assist each other with debugging problems. We did not encourage pupils to 'pair up' routinely at computers, since there was a danger that some pupils might not participate fully in the work.

4.5.3 *How the learning materials were utilized*

All of the learning resources developed within the programming project (see 3.3.3) were used. The task sheets, help hints, design grids and log sheets were available at the resource area, and pupils were given the answer booklets, summary sheets and revision questions to keep in their folders. The answer booklets were corrected by the class teacher as soon as possible after each pupil completed a topic, and she would make written comments against the pupil's work or on their progress records on any corrections or additional work that needed to be done. She endeavoured to ensure that pupils did eventually complete each topic to a satisfactory standard, even if it meant that in the short term some pupils were held back from making further progress until they had put right any shortcomings in their work, such as missing documentation or

errors. The revision questions were done as homework whenever the pupil reached the end of a topic.

4.5.4 *Creating an appropriate climate for learning*

The success of the learning environment would depend crucially upon the creation of an appropriate classroom climate. Pupils should feel confident about expressing their opinions honestly and openly in the knowledge that their opinions would be valued by others. Moreover, the social climate should promote critical appraisal and reflection, thus enabling learners to reach a better understanding of the factors which facilitate or impede their learning.

Resnick (1987) highlights the importance of the social setting to cultivate the disposition to engage in higher order thinking. She warns against a 'quick fix'; there is a requirement for sustained long term cultivation of higher order thinking in order for pupils' perceptions of the value of thinking and reaching independent judgements to grow, and for them to feel secure about expressing their opinions openly. The case study timescale accommodates this.

The self-paced and individualized approach places greater responsibility on pupils to monitor and manage their learning, in comparison to more traditional 'chalk and talk' teaching methods. The teacher does not begin the lesson by reminding all pupils of the work that they did in the previous lesson, and nor does he/she summarize what has been covered at the end of the lesson. It is up to the pupils individually to look back and review their learning on a day-to-day basis, and through doing so consolidate their knowledge and develop their ability to solve problems (Polya, 1948). Pupils must monitor their pace of learning, their progress with solving the problems in hand, and their understanding, taking appropriate actions to remedy difficulties. The teacher's pedagogical interactions with individual pupils or small groups should be designed to support pupils in these processes.

4.5.5 *Pedagogical interactions*

The framework for the teacher's pedagogical interactions, which is set out in the notes for teachers on unit 1 (p.10), was adapted from Perkins and Martin's (1986) experimental design. When a pupil (or it could be a group of pupils) either seeks help from the teacher, or it is clear to the teacher that the pupil has reached a stage of needing help, there are a number of possible recourses. The teacher's first recourse is generally to refer the pupil back into the materials to re-read an explanation or do some form of checking (for example, is the error covered in the help hints?). The second recourse is to prompting (for example,

'What did the design for your last program look like?'). The third recourse is to hinting (for example, 'Have you missed something out?' or 'What does this command do?'). The fourth recourse is to providing a solution and explanation to the immediate dilemma to enable the pupil to continue with the task. The fifth recourse is to providing more extended assistance (such as coaching or remediation). This advice should not be adhered to rigidly; the teacher should exercise his or her professional judgement about how and when, precisely, to intervene.

The intention behind this strategy is to maximize the opportunity for the pupil to come up with most of the solution for himself or herself, in accordance with Polya's (1948) advice on giving unobtrusive help. Perkins and Martin describe prompts as the high level strategic questions that one might ask oneself concerning several aspects of problem solving such as formulating goals, generating solutions, and evaluating solutions. The notes for teachers on unit 1 (p.10) recommends the following sequence of prompts when a program runs but does not produce the expected output:

> When a program runs but does not produce the expected output, this sequence of prompts is recommended: "What did the computer do?"; "Is this what you wanted it to do?"; "What does your program tell the computer to do?"; "What changes do you need to make to your program?".

Clements and Gullo (1984) used a similar set of prompts to assist students doing Logo programming (see 2.7.3). The teacher's use of a simple sequence of prompts such as this provides a *model* of how to respond in situations where a problem has arisen, which the learner should then internalize. This research design is, unfortunately, not ideally suited to gauging in a direct way the success of this strategy, since it would be necessary to sample episodes of learning as they occurred in order to determine whether pupils had self-prompted spontaneously (and even then it might not be all that obvious from observing pupils' actions). Both the class teacher and I adopted these approaches in our interactions with pupils, and pupils were also encouraged to do likewise when assisting each other. As pupils' programming expertise developed, we would sometimes direct pupils to other pupils for assistance.

4.6 METHODS OF DATA COLLECTION

The case study is based on a combination of data gathered throughout the study. The following sub-sections outline the different sources that were used: (1) records of pupils' individual rates of progress, agreed targets, attendance, and summative grades; (2) folder of work in programming; (3) assessment

portfolio; (4) pupil questionnaires; (5) written review tasks; (6) interviews with pupils; (7) classroom observations and interactions with pupils; and (8) discussions with the class teacher.

4.6.1 *Records of pupils' individual rates of progress, agreed targets, attendance, and summative grades*

The class progress record and summary class progress record were used to record individual rates of progress and attendance, and they were kept up-to-date by the class teacher. Progress targets were negotiated with each pupil at the start of the summer term in S3 to cover the remainder of the term, and again at the start of S4 to cover until mid-October. This process was delayed until the programming course was well under way to allow decisions to be informed by each pupil's progress up to that point, his/her aspirations, and the class teacher's judgement of the pupil's capabilities. Each pupil should have maintained the progress record at the beginning of the answer booklet, noting down the point reached by the end of the period and anything important to remember to do at the start of the next period.

Evidence from the various progress records is discussed in chapter 5 and from the assessment record (indicating summative grades) in chapter 6.

4.6.2 *Folder of work*

The folder of work contained the pupil's completed answer booklets, any printouts, design grids and answers to the revision questions. Although pupils retained all of their finished programs as files saved to their own disc areas and as printed listings, they were not asked to retain their earlier, 'buggy' programs, as it was felt that this would lead to confusion between the different versions and spending too much time on obtaining printouts and organizing these in their folders.

The evidence from the folder of work is used selectively by examining pupils' written solutions (in the answer booklet) and printouts for topics 9 and 10 (see chapter 5).

4.6.3 *Assessment portfolio*

As the pupil completed each programming coursework task, it was marked by the class teacher and then retained in the pupil's assessment portfolio. The pupil received individual feedback on his/her performance in the task. At the end of the course the class teacher and pupil selected the two items that would be used as evidence towards the programming coursework grade. The basis for

Closed response items in which pupils were asked to assess:

- how interesting they found the work to be in each unit;
- their ability to understand what was being taught within each unit;
- their ability to follow the instructions in each unit;
- their confidence about various aspects of the work (including any aspects they had indicated they were unsure of in the previous questionnaire);
- how useful the help hints were (Unit 1 only);
- whether they liked working at their own pace (Unit 1 only);
- how easy it was to get access to a computer (Unit 1 only);
- whether they were pleased with their progress (Units 2–3);
- whether they were looking forward to doing the next unit (Units 1–3);
- whether they would like to learn more about programming (Unit 4).

Open response items in which pupils were asked to:

- identify any programs that they liked in each unit and say why they liked them;
- identify anything that they didn't like about the work in each unit;
- provide information on their prior experiences of programming before embarking on Unit 1.

Items in which pupils were asked to explain their choice on...

- the effectiveness of particular resources (Unit 1 – answer booklet, Unit 2 – revision questions and summary sheets, Unit 3 – ideas to explore);
- whether they preferred to work on their own or with a partner (Unit 1);
- whether they considered that it had been worthwhile to learn to program (Unit 4).

Figure 3: Pupil questionnaires – description of items.

the selection of particular tasks is described in chapter 6 where the findings on programming attainment are examined. The assessment portfolio also contained a range of non-programming coursework tasks and a project, each based upon practical problem solving using a general purpose package.

The evidence from the assessment portfolio is used in two ways, mainly to determine the level of pupils' programming attainments by the end of the course, but also to examine (near) transfer effects from programming by taking the spreadsheet items as examples of computer problem solving using a general purpose package. This transfer performance has been selected because it is authentic, and the scheduling of topics on the course ensures that pupils will have completed or almost completed their programming work by the time they begin on spreadsheets.

As with the folder of work, the evidence is used selectively. Each pupil's solution to one programming coursework item (one of the two tasks selected as evidence towards the programming coursework grade) is examined, and an overview is taken of the class's performance in programming coursework. To assess near transfer, an overview is taken of the class's performance in a spreadsheet coursework task and project. (Pupils' verbal comparisons between learning to program and learning to use a spreadsheet, and between the problems and solution methods encountered, are captured in the interview data.) The findings on near transfer are discussed in chapter 8.

4.6.4 Pupil questionnaires

Pupils' affective responses to learning to program were gauged through the end of unit questionnaires. These enabled data to be gathered at interim stages of the study. For any pupils who had not finished the unit they were working on when the course ended, the questionnaire was adapted so that only relevant questions were asked to ensure that there was an opportunity to review recent work, but this data was not collated.

The questionnaire items were in one of three formats: closed response (select from given options); open response; or where the respondent was asked to choose an option and also to give a reason for his/her choice. The items were wide ranging to encompass pupils' self-appraisals (on interest, confidence, ability to understand etc.), learning preferences (e.g. on working individually or with a partner), and appraisals of the learning resources and environment (e.g. 'Did the revision questions help you to understand the work?', 'Could you get access to a computer whenever you needed to?'). A description of the items is shown opposite (figure 3).

The questionnaire responses were used as the main starting point for conducting individual or group interviews. From the manner in which the end-of-unit questionnaires had been completed by other pupils in the two sessions prior to the commencement of the case study, I was confident that the responses would provide a satisfactory basis for further discussion. The use of follow-up interviews was designed to overcome some of the recognized limitations of questionnaires, i.e. that the information collected tends to be descriptive rather than explanatory, and even when written explanations are sought they are often brief and superficial (Munn and Drever, 1990). Although both written and spoken modes of communication are subject to bias towards those who are more articulate, pupils do generally find it easier to communicate fluently using speech, and they can be persuaded through the use of non-directive probing to clarify and elaborate on their statements (Brown and McIntyre, 1993).

Pupils did not complete the questionnaires anonymously to enable questions during interviews to be targeted to individual pupils. Also pupils were not assured of confidentiality (except beyond the classroom walls), since otherwise group interviews, which were intended to facilitate open dialogue, could not have taken place (unless the questionnaire data played no part in the interviews or pupils were left to volunteer any information that they wanted to share). The class teacher may also have wished to follow up sympathetically with individual pupils any concerns expressed (such as anxiety about falling behind with the work).

However, when denying pupils the opportunity to express their views privately, one must be aware of the possible consequences for individuals and for the research outcomes. There is clearly a responsibility to conduct the research in a manner which avoids situations where a conflict of interests, embarrassment or hurt may result, and to deal sensitively with any incidents that do arise. Pupils as research participants are particularly vulnerable as a consequence of the school's authority structure. The class teacher's and my own status could make some pupils reluctant to express their opinions honestly. They may fear that they would be disadvantaged if their responses, or refusal to participate, displeased us, or else they may be motivated by the desire to gain adult or peer approval by responding in a manner that they thought would bring this about.

I decided to monitor the situation carefully. There was no evidence from any source that pupils answered dishonestly, although occasionally during interviews pupils were reticent or unable to give a response (it was sometimes

difficult to distinguish which)[13]. Nearly all pupils seemed to welcome the responsibility that was accorded to them as participants in the research, helping to create effective conditions for learning within their own classroom, and contributing to the longer term goals of the research (this is brought out more fully in later chapters).

The evidence from questionnaires is reported in chapter 7 and referred to occasionally in other chapters where it is relevant.

4.6.5 Written review tasks

There were three written review tasks, the findings from which provide evidence on pupils' reflections on learning, problem solving and programming.

The first review task was designed to be completed by each pupil individually on reaching the end of a unit. It involved writing up a favourite problem(s) – What had you to do in this problem? Why did you like this problem? What did you learn from this problem? – and then pasting a printed listing and sample run of the program beside this. This task enabled pupils to expand on the response to a questionnaire item ('Which programs did you like the best? Say why you liked them.'), thus prompting a more in-depth review. Pupils were asked to do the written part of the task at home.

The second review task focused on problem solving strategies and was carried out by a sample of pupils. It asked each pupil to identify where he/she had used each of eight different problem solving strategies within unit 1, (e.g. 'Can you show me somewhere, in unit 1, where you had to: (a) starting with a new problem, break it down into a number of smaller problems? (b) ...'). There were three other questions that concerned, in turn, the usefulness of the strategies that had been taught for solving programming problems, the pupil's awareness of having tackled problems in other situations in similar ways, and whether anything that had been learned in the first programming unit might facilitate other learning.

The third review task was completed at the end of the course, following interviews. I asked pupils to discuss in small groups and then for each individual or group to enumerate at least ten points of advice that would assist

---

13 There was some evidence that pupils had difficulty with recollecting events *accurately* during interviews, particularly in relation to the earlier topics within each unit. This was not surprising because of the time that it took some of them to complete the unit, and also because of school holidays and other interruptions to their study.

Interview	Pupil(s)	Date(s)	Based On	Length (mins.)
Phase 1				
1	Chris	15/10/93	Unit 1	25
2	Lorraine	15/10/93	Unit 1	25
3	Paula	1/11/93	Unit 1	45
4	Claudia	9/11/93, 23/11/93	Unit 1	45
5	Scott	29/11/93, 30/11/93	Unit 1	25
Phase 2				
6	Chris, Lorraine	22/2/94	Unit 2	45
7	Paula, Kathleen, Caroline	9/5/94	Unit 2	45
8	Claudia, Kirsty, Kelly	10/5/94	Unit 2	45
9	Catherine, Bernadette, Claire	24/5/94	Unit 2	45
10	Carol-Anne, Bryan, Scott	14/6/94	Unit 2	45
11	Dawn, David	20/6/94, 21/6/94	Unit 2	45
Phase 3				
12	Dawn, Bernadette	18/11/94	Topics 11–13	25
13	Gerard	2/12/94, 9/12/94	Topics 11–12	40
14	Paula, Chris, Kirsty	20/12/94	Topics 11–17	60
15	Catherine, Kelly	13/1/95	Unit 3	25
16	Scott, Bryan	20/1/95	Topics 11–13	25
17	Claudia, Kathleen	27/1/95	Unit 3	25
18	Anthony	13/2/95	Topics 11–12	40
19	Michael	14/2/95	Topics 11–12	25
20	Stephen	28/2/95, 3/3/95	Unit 2	55
21	David, Claire	6/3/95, 7/3/95	Unit 3	55
22	Lorraine	9/3/95	Topics 11–16	50
23	Paula	9/3/95	Topics 11–17	40
24	Carol-Anne	10/3/95	Unit 3	25
25	Caroline	20/3/95, 27/3/95	Topics 11–16	50
Unit 1 – Topics 1–5; Unit 2 – Topics 6–10; Unit 3 – Topics 11–14; Unit 4 – Topics 15–18.				

Table 5: Interviews with pupils.

a beginner programmer (for example, someone just starting the course) to produce good programs in an efficient manner. What would constitute a good program and an efficient manner was left deliberately for pupils to unpack. The intentions behind this task were, firstly, to provide a meaningful context for pupils to reflect on their learning experiences and to share their insights with each other, and secondly, to provide a summative measure of pupils' insights about learning, problem solving and programming.

Chapter 8 provides a full specification of review tasks 2 and 3 and an indication of the manner in which each review task was used within the case study, in addition to presenting and discussing the findings.

4.6.6 Interviews with pupils

4.6.6.1 The pattern of interviews, and interview procedures

There were three phases of interviews (see table 5), in the first term of S3 as most pupils reached the end of unit 1, in the spring and summer terms of S3 as most pupils reached the end of unit 2, and between October and March of S4 as all pupils neared the end of their programming work. Not all pupils were interviewed on completion of every unit since this could not have been accommodated within the available time. The actual pattern of interviews is explained in the paragraphs below.

Pupils were usually withdrawn from a programming class to be interviewed in order to minimize any disruption to other aspects of their school work. It was very difficult to find a quiet location where pupils would be at their ease, and various locations were tried, including another classroom, the staff base, the school library, and even the corridor outside the classroom. The library proved to be the best location, in spite of the chatter in the background, because pupils seemed most relaxed there. The seating arrangement was around a table, at adjacent sides (not opposite me; if there were three interviewees one was seated on one side of me and two on the other). It was possible with this arrangement for pupils' voices to be picked up on the tape recorder, for me to maintain eye contact with each interviewee, and to examine resources (such as the task sheets for the unit we were discussing) together, if this was necessary.

The timetabling of interviews in advance also proved to be very difficult. In addition to scheduled interruptions – school holidays (which frequently coincided with the days on which the class was held), exam diets, a work experience placement – there were, of course, pupil absences. Making accurate predictions of when pupils were likely to complete a unit was virtually

impossible, until they reached the stage when they had almost completed it. The Phase 3 interviews were most difficult to fit in because pupils had less timetabled time for programming in S4.

The first phase of interviews was restricted to five pupils who were interviewed separately on completing the first unit. Individual interviews were conducted in this phase rather than group interviews because of their exploratory nature. It was possible to interview only a sample of pupils since the class's progress had not begun to separate out at this early stage of the course. Thereafter pupils were interviewed either in small groups or individually.

The duration of interviews varied. One of the reasons for this was to enable the interviews to be conducted in an unhurried manner, so that less articulate pupils would not feel under pressure to respond quickly, and to allow them time to clarify and expand upon their answers. The interview would be continued during the next programming lesson if time ran out. In situations where pupils had some difficulty with recollecting their earlier work, some time was spent on looking back over the unit and discussing the work within it. Group interviews usually took longer than individual interviews, as did interviews which were based upon work covering more than one unit (e.g. interviews 14 and 25).

Pupils were asked if they would agree to the interview being recorded on cassette tape. Claudia did not want her first interview (interview 4) to be taped, however she did agreed to notes being taken by me during the interview. These were typed up immediately and she was asked the next day to verify them. She did agree to her later interviews being taped. Unfortunately interview 8 did not tape correctly because the recorder had been on the wrong setting, and notes were typed up immediately following the interview and verified for accuracy and completeness by the pupils the next day. These notes were less detailed because they had to be constructed from the set of questions prepared before the interview and from recollection of what was said during it. (I did not attempt to make notes during group interviews as this would have been very difficult to do and it would have slowed down the interview considerably.) There was one further interview arranged with Paula (interview 23) because her voice had been very indistinct on the recording of her previous interview.

Having consulted with the class teacher, I decided that it would not be reasonable to ask all pupils to verify the accuracy of the tape transcripts because

of the time that it would take pupils to do this, and because there would be unavoidable time lapses between interviews taking place and the transcripts being verified (for example, as a result of school holidays). To compensate for this, I double-checked each transcript against the recording.

4.6.6.2 Phase 1 interviews

In the first phase, five pupils – three girls and two boys – were interviewed separately shortly after finishing unit 1. This set of pupils was selected to reflect the gender balance and to exhibit a range of responses to the learning environment (according to their questionnaire responses). The class teacher also made an input to the selection. Unfortunately some pupils did not complete the unit in time to be interviewed during this phase.

The interviews were semi-structured and used a mixture of closed and open questions. They were not conducted too formally; they were certainly less formal than Drever (1995) recommends. In addition to my initial questions, prompts (to encourage responses) and probes (for the answer to be developed), there were occasions when I attempted (not always successfully!) to remedy pupils' misunderstandings when these emerged in discussion or when pupils specifically asked for an explanation, generally in relation to an aspect of the work within the unit that we were discussing at the time. It was appropriate to do this since the interviews were intended to provide a context for pupils to review their learning and to strengthen their understanding as well as being an important research tool.

As a way of settling into the interview, the pupil was asked to discuss the program(s) that he/she had chosen to write about in the first review task. Then the pupil's responses to the questionnaire were gone through one by one. For example, when one pupil had written, in response to an open item inviting comments about any disliked aspects of the work, that she did not like using or writing out long programs, I asked: (1) 'Which programs were too long in the unit?'; (2) 'Why didn't you like using or writing them out?'; and (3) 'Do you think you would have got on better if you didn't have to write the program out before going to the computer?'. Some questions were prepared in advance from the questionnaire responses and others arose naturally in the discussion. A similar approach was adopted to discussing the second review task which focused upon problem solving strategies. This set the pattern for subsequent interviews. Two advantages of building the interviews around the questionnaire responses and written review tasks were that pupils had the

opportunity to give prior consideration to the issues, and they were not caught 'off balance' by being asked unexpected questions.

4.6.6.3 *Phase 2 interviews*

The second phase of interviews involved sixteen pupils who were interviewed in small groups. The timing of interviews coincided as closely as possible with each pupil completing the second unit. Four boys (Gerard, Anthony, Michael and Stephen) did not complete the unit on time to be interviewed in this phase.

There were a number of reasons for conducting small group interviews, in addition to the most obvious reason of enabling most pupils in the class to participate. Group interviews would hopefully promote a relaxed atmosphere in which pupils would feel less inhibited. Also, Drever (1995) suggests, group interviews can capture the normal patterns of interaction within a group, which may result in better evidence (I tried to form the groups from pupils who collaborated and interacted frequently[14]). However the most important reason was to enable me to mediate the process of pupils discussing together the learning and problem solving strategies that they had used on the course, and through this gaining better insights into their own thinking and learning.

The advantages of conducting group interviews had to be weighed against potential drawbacks. The main anxiety was that some pupils might be unduly influenced by other pupils' responses and would not state their own opinions; on balance this seemed unlikely since pupils had already completed the review tasks and questionnaires individually, and my initial questions were based on their individual responses. A further anxiety was that some pupils might not contribute very much to the discussion, and for this reason the group size was restricted to a maximum of three, and some questions were directed to individual participants rather than being fielded openly.

4.6.6.4 *Phase 3 interviews*

In the third phase interviews were conducted involving all of the pupils in either individual or small group settings. The four boys who had not been interviewed in the summer term of S3 were each interviewed separately to permit a more in-depth discussion of their most recent work.

---

14 Pupils were informed individually of the name(s) of the other pupil(s) in their group when gaining their consent to be interviewed. I indicated that they could, if they preferred, be interviewed on their own, but no-one opted for this.

Parts of the phase 3 interviews were very exploratory, for example, discussions focused on a range of planning and debugging strategies and their counterparts across the curriculum, how one could improve the design of the user interface for some programs, or how using a general purpose programming language such as COMAL compared with using a general purpose package. Some of the interviews probed very deeply, for example, the discussion, related in chapter 7, of Stephen's difficulties with learning programming (from interview 20), or the discussion, related in chapter 8, of Caroline's insecurities about problem solving (from interview 25). The intention was to stretch pupils' understandings in addition to providing further data. There was thus generally a greater variety of questions than in previous interviews, and the questions varied more from pupil to pupil. While I was aware that this might restrict opportunities for comparisons in the data, on balance I felt that it was worthwhile in order to gain better insights on individual pupils, and that it was also necessary to accommodate the individual differences among them which had become evident over the duration of the study.

4.6.6.5 Summary

All of the interviews were designed to play an important role in the creation of a powerful learning environment (De Corte, 1990) by making students more explicitly aware of their own cognitive and metacognitive activities through processes of articulation, reflection and exploration of ideas. They were designed to support pupils in learning complex subject matter by strengthening and extending their understanding, giving them greater confidence, and providing encouragement. They should also provide a stimulus for higher order thinking by engaging pupils in reasoning, justifying, analysis, enquiry and critical appraisal.

The interview data is integrated into the findings and discussions in chapters 7 and 8, which focus on pupils' affective responses to learning to program and their reflections on learning and problem solving.[15]

4.6.7 Classroom observations and interactions with pupils

In addition to the time spent on interviewing, I spent thirty hours in the programming class assisting pupils when they requested or needed help, and checking their work when pupils wanted this done. This manner of interaction enabled me to get to know pupils individually, it helped to reduce

---

15 Some editing of the interview data has been done in the thesis to reduce its length. The transcripts have not been edited down.

any barriers that may have led pupils to feel awkward or embarrassed when interviewed, and it informed the interview discussions. Engaging in pedagogical interactions with individual pupils or small groups also provided an additional means of gauging pupils' understanding and competence, and it had the considerable merit of enabling the class teacher and I to combine resources to address more effectively the learning needs of all pupils in the class (Simpson and Ure, 1993). I also joined the class on a few occasions as they did practical problem solving using a spreadsheet or database to enable me to observe the teaching processes.

I did not attempt to make detailed and systematic observations of lessons by using an observation schedule. Such a record would have contained discontinuities since I was not present during every lesson[16], and it would have forced me into a more distanced role in the classroom.

4.6.8 Discussions with the class teacher

Throughout the study regular, informal meetings took place between the class teacher and I to plan and co-ordinate the research, and to discuss the findings as they emerged. Pupils were made aware that we did discuss all aspects of their work, and that we viewed their contributions in terms of the opinions that they expressed within questionnaires and during interviews as essential to the creation of an effective learning environment. It was hoped that this constructive emphasis would help to alleviate any fears that pupils may have had about expressing negative opinions.

I did not attempt to capture these discussions for the purposes of reporting them in the thesis. The class teacher would not have wished this to happen, and it would probably have hampered the discussions to attempt to do this. Suffice it to say that this study would not have been possible had it not been for the willingness of the class teacher to support it wholeheartedly.

4.7 DESIGN OF EVALUATION

The overall success of the learning environment in the case study class has to be gauged against two sets of criteria which must be jointly considered, relating firstly to the intrinsic qualities of the teaching processes, and secondly to the quality of the intended educational outcomes in terms of pupil learning

---

16 The class teacher could also have been asked to contribute her written observations in order to compile a more complete record, but I did not consider this to be the best use of the time that she could devote to the study.

Case study setting – intended educational outcomes

Did the learning environment enable pupils to:

1. develop an understanding of programming concepts, principles and techniques?
2. develop competence at solving programming problems?
3. develop a range of higher order thinking skills?
4. develop metacognition and 'learning to learn'?
5. develop the ability to reflect on the broader applicability of problem solving processes and strategies?
6. demonstrate near transfer from programming to practical problem solving using a spreadsheet?
7. develop positive affective responses towards learning to program?

Figure 4: Case study setting – intended educational outcomes.

Sources / Criteria	folder of work	assessment portfolio	question-naires	written reviews	interviews
1. understanding	M	M			M
2. competence	M	M			
3. higher order thinking skills	M	M		M	M
4. metacognition/ learning to learn	M	M	M	M	M
5. reflection				M	M
6. near transfer		M			
7. affective outcomes			M	M	M
NOTE: M – indicates a major contribution.					

Table 6: The contribution of data collection methods to the case study evaluation.

(Elliott, 1991). The intended educational outcomes are broadly stated in the form of evaluation questions (figure 4). The intrinsic qualities of the teaching processes, which have been discussed at length in earlier chapters, will be considered as an equally important aspect of the findings in relation to these outcomes.

The manner in which each of the five main data collection sources (folder of work, assessment portfolio, pupil questionnaires, written reviews and interviews) will contribute to this evaluation is indicated in table 6. All of these sources make some contribution to the evaluation of each area (except in relation to criterion 6).

Since all of the criteria are highly inter-related, they are very difficult to tease out in the data. It is nevertheless helpful to be able to place a spotlight on a particular criterion when analysing the data and drawing conclusions.

The data will be analysed to identify general trends (such as the overall reactions of the class to aspects of the learning environment, and gender differences) and also to compare and contrast the responses of individual pupils to the learning environment across a range of dimensions. The use of a range of instruments will allow the data to be triangulated, which is a method of bringing different kinds of evidence – observations or accounts of a situation collected from a variety of angles or perspectives – into some relationship with each other so that they can be compared and contrasted (Elliott, 1991).

The case study findings are presented and discussed as follows: on progress, using the evidence from progress records and the folders of work (chapter 5); on programming attainment, using the evidence from the assessment portfolios (chapter 6); on pupils' affective responses, using the evidence from the questionnaires and related interview findings (chapter 7); on pupils' reflections on learning and problem solving, using the evidence from the review tasks and related interview findings (chapter 8); and on near transfer, using the evidence from the assessment portfolios and related interview findings (also chapter 8).

4.8 REVIEW AND DISCUSSION

The review and discussion focuses upon four issues: the research design and comparisons with other research designs; generalizing from case study evidence; factors affecting the interpretations of the data; and an examination of my role in the research and whether it could be replicated by others.

4.8.1 *Discussion on the research design and comparison with other designs*

By focusing on learning in progress, the case study seeks to capture pupils' emerging understanding and competence as they grapple with a new and complex subject matter over an extended period. Through a detailed study of this nature, one can establish that the requisite learning opportunities are being provided, and one can seek to put in place a range of valid and reliable measures that reflect the many facets of pupils' learning (Harlen, 1994).

The case study thus avoids reliance on end measures of attainment, or on pre- and post- test comparisons. Two practical difficulties associated with testing as a means of evaluating new materials or innovative approaches are that it is often difficult to design assessments which encompass all of the intended learning, and some aspects of learning are difficult to test by conventional means (e.g. different ways of thinking and solving problems, attitudes and perceptions). Tests which emphasize the reproduction of factual information can suppress efforts to teach problem solving and critical reasoning, since examinations may have a strong influence on what is taught, how it is taught, and how students set about their learning (Nisbet, 1990; Resnick, 1987).

Harlen (1994, p.1) raises further theoretical objections to approaches that rely solely on testing:

> The theoretical objections can be realised by supposing, for the sake of the argument, that we do have valid and reliable measures of learning. This would tell us what pupils had learned but not whether the learning was as much as could be expected, nor whether it resulted from the use of the materials and not something that would have happened anyway. The suggestion of pre-testing and post-testing and using trial and control pupils simply compounds the practical problems. It is now widely accepted that we cannot pin down cause and effect using an experimental design as in science; there are just too many differences between pupils for this to work and 'random' allocation of pupils to groups would interfere unacceptably with inter-personal relationships which are important in learning.

In order to contextualize some of Harlen's objections to research methods based upon testing and experimental designs, I shall refer to her criticisms in a recent review of research on ability grouping (Harlen and Malcolm, 1997) of research methods used in many classroom based studies.

- Confounding of different effects inherent in the major question they address. (If the question is, 'how does grouping by ability affect learning?', in practice pupils are not put into groups or classes by ability and then treated in exactly the same way, which some would argue is pointless and

others would argue is impossible. It is never just the effect of grouping that is being compared but also the impact of different teaching methods, the quality of teaching, teacher expectations, pupil expectations and sometimes different materials.)

- The relevance of the measures used in quantitative studies. Where there is a difference in the content or methods used with each of the groups being compared it is clearly important that the measure should reflect the learning experience of both equally.

- Attempts to 'control' the treatment of groups being compared are doomed by the teacher variable. Teachers' attitudes and dispositions affect their willingness and ability to put in place a particular teaching approach.

- The vast majority of studies lack information of the kind that would enable conclusions to be drawn (in this instance, about how or why the composition of groups does or does not make a difference) because of their over-reliance on outcome measures.

Within this study, it would not have been feasible or desirable to use a pre-test to measure pupils' understanding of programming concepts, principles and techniques (criterion 1), their competence at solving programming problems (criterion 2), or their affective responses (criterion 7) since the majority of the class had no prior experience of programming beyond a very brief introduction in S2. (Imagine, as a pupil on your first day in the programming class, being given a test in which you do not recognize most of the questions and cannot even begin to answer them.) In terms of pupils' reflections on the broader applicability of the problem solving strategies that they utilized during the programming course (criterion 5), pupils need to have encountered these strategies first to be able to assess their usefulness. They may have given little thought in the past to their 'method of learning' (Nisbet and Shucksmith, 1984, p.5) and hence even the language of learning may be unfamiliar to them, thus hampering meaningful discourse.

The extent to which the learning environment stimulates pupils to engage in higher order thinking (criterion 3), and fosters metacognition and learning to learn (criterion 4) is gauged in an on-going manner through analysis of pupils' performances (practical, written, spoken) in relevant tasks, enabling a more nuanced judgement to be reached of the influence of the learning environment on the quality of each pupil's thinking and learning. One must also be aware of the other significant influences on pupils' intellectual development during the course of the study, such as their increasing maturity,

the formal and informal curriculum, and their everyday lives and relationships. Within the formal curriculum, problem solving, investigation and higher order thinking feature widely across Standard Grade courses, although their embodiments differ from subject to subject (Holroyd, 1989) and there is little guidance within the syllabuses on how they should be learned or taught (Nisbet, 1990; Drever, 1988). One can presume, therefore, that there will be considerable variation among pupils in how they experience problem solving, investigation and higher order thinking across the range of subjects they are taught over the duration of the study. It is clear that end measures and pre- and post-testing are too crude to be useful in this context.

The use of a comparative study, in which the effects of different teaching methods are measured and contrasted, was ruled out for a number of reasons, the most fundamental being the ethics of assigning pupils to different 'treatment' groups. This was felt to be particularly inappropriate at this stage of their education with the approach of Standard Grade examinations, the outcomes of which affect their future prospects. Harlen's observations on confounding effects and the many differences between pupils and teachers which hinder comparisons, also come into play.

Papert (1994, p.26–7) expresses eloquently his objections to studies based upon the 'scientific method' when he highlights the unique, personal significance of events that shaped his own intellectual development.

> Researchers, following the so-called scientific method of using controlled experiments, solemnly expose children to a "treatment" of some sort and then look for measurable results. But this flies in the face of all common knowledge of how human beings develop. Although it is obvious to me that my newspaper played a profound role in my intellectual development, I am pretty sure that no test would have detected its role by comparing my "performance" the day before I started and three months later. The significant effects emerged over a much longer period, to be measured, probably, in years. Moreover, an experiment that gave a hundred children "the experience of producing a newspaper," even if continued for several years, still would miss the point of what happened to me. The significant engagement was too personal to be expected to operate as a mass effect. I fell in love with my newspapering (as I did with mathematics and other areas of knowledge) for reasons that are as personal and in a sense as unreproducible as those that determine any kind of falling in love.

Papert's observations present a considerable challenge to educators and educational researchers. At the very least, Papert points to the need to take account of individual differences when analysing research findings, and to focus on both affective and cognitive outcomes. More profoundly, he suggests

that learning environments must be designed flexibly to aid personal appropriation so that all children have the opportunity to experience the intellectual excitement that he experienced through his (self-devised) newspaper project. Unfortunately the constraints of examination syllabuses and the pace of learning at S3 to S6 can make it especially difficult to achieve this.

Kafai's (1993) study provides a very good example of a learning and research environment which is designed to accommodate individual differences. Her detailed profiles of three computer game designers provide rich evidence to supplement her findings on the whole class. I considered carefully whether it would be useful to build individual pupil profiles within my own research, however I decided against this approach for the following reasons: (a) in comparison to Kafai's study, it would be much harder to chart the progress of individual pupils across a range of activities, as opposed to one extended activity, each of which would have to be accurately depicted – this would add unacceptably to the length of the thesis; (b) it would be difficult to map out a continuum in relation to important aspects of pupils' learning with only a small sample of pupils on whom any detailed evidence is presented; (c) it avoids a selective focus which could distort the evidence; and (d) there is the very real dilemma of which cases to choose from amongst all the interesting cases. Instead the evidence on individual pupils is fully integrated into the discussion of the findings.

I also considered whether it would be useful to compare certain of the findings from the case study class with the equivalent findings from some other classes coming 'on stream' in August 1993 as part of the field testing of the later study units within the programming project. For example, the data from class records and questionnaires could be collated and compared in various ways. The purpose would be to locate the case study class within a wider context rather than to conduct an experiment. I decided that the exercise would not be illuminating. The SOED report (Kirkwood, 1996) indicates the enormous variations in teachers' practices, for example, in the way programming classes are scheduled and the time allocated to the programming topic, and also some very large variations in class compositions in terms of gender balance and attainment spread. However it is worthwhile to note briefly that the learning environment evaluated very well from the evidence gathered from pupil questionnaires across seven different classes.

4.8.2 *Generalizing from case study findings*

An essential feature of case study research is that its focus is an entity of intrinsic interest, not merely a sample from which to learn about the population, and that the choice of methods is related to the nature of the case as well as to the purposes of the study (Simons, 1989). Hence one cannot assume that the research method can be replicated precisely, or that findings can be immediately generalized. There are close parallels here to teaching. When considering the implications of their investigation of how teachers construe their teaching, Brown and McIntyre (1993, p.113) summarize as follows:

> The professional craft knowledge which teachers have revealed to us is highly complex, providing no simple generalizations about how to do anything well in teaching.

However Elliott (1991, p.65) points to the capacity of reflective teachers to generalize from past experience:

> Reflective teachers would do well to engage in a little meta-reflection on how they deliberate about what to do in a particular situation. ... They develop their understanding of the present case by discovering the ways in which it is similar to and different from other cases in their experience. In this process teachers assume they can generalize from past to present experience. Then why not also assume that other professionals' case studies can provide vicarious experiences which are generalizable to their own situations, and vice versa?

He argues that the assumption that qualitative case studies are low in generalizability arises from the empiricist tradition in which generalizability depends upon the extent to which data can be statistically aggregated.

Taken together, the views of Brown, McIntrye, and Elliott would suggest that experienced teachers should be very well placed to assess the significance of case study findings for their own teaching situations, provided that the case study environment is sufficiently well documented to enable its relevance to be assessed, and valid comparisons and inferences to be drawn. For this to happen, findings must be disseminated widely (beyond library shelves) and accessibly.

In terms of how one assesses the significance of research findings on computers in learning, Papert warns against narrow, literal-minded approaches. He draws an analogy to the first successful airplane flight.

> The year 1903 – when a powered airplane first flew successfully – was a turning point in the history of transportation. But the famous flyer made by Wilbur and Orville Wright did not prove itself by its performance. The duration of the best of several flights that day was only fifty-nine seconds. As a practical alternative to the horse-drawn wagon, it was laughable. Yet imaginative minds could see in it the birth of the industry that would lead to the jumbo jet and the space shuttle. Thinking about the future of education demands a similar labor of the imagination. The prevalent literal-minded, "what you see is what you get" approach measuring the effectiveness of computers in learning by the achievements in present-day classrooms makes it certain that tomorrow will always be the prisoner of yesterday. Indeed, the situation in education is often worse than judging the effectiveness of airplanes by the fifty-nine-second flight. It is more like attaching a jet engine to an old-fashioned wagon to see whether it will help the horses. Most probably it would frighten the animals and shake the wagon to pieces, "proving" that jet technology is actually harmful to the enhancement of transportation. (Papert, 1994, p.29).

A further point that can be taken from this is that one cannot expect educational innovations to work in unfavourable circumstances (this echoes Harlen's more specific points about the timing of innovations).

4.8.3 Interpretations of the data from this study

There are four issues affecting the interpretation of the data which I wish to explore briefly. These are the status of the data, difficulties with establishing clear communication, participants' potentially inaccurate, incomplete or distorted accounts of events or reconstructions, and potential gender bias.

4.8.3.1 The status of the data

Drever (1995) reminds us that pupils' responses during interviews (and this would also apply, in this research, to questionnaire and review task responses) are, 'affected by expectations derived from the classroom (is this a sort of test? are they supposed to know the answers?) and notions of what it is proper to say to teachers.' He adds, 'In interpreting the results, you need to bear in mind that what you have got is not "pupils' views" so much as "statements made by pupils in an interview"... ,' (p.52) but he cautions against attempts to make allowances for this and against reinterpreting what they have said. My response to this situation was to explain to pupils why I wished to interview them, both at the time when I sought their verbal consent and at the start of the interview. I told them that it wasn't a test, that it didn't count towards assessment (the letter which went to parents also gave these assurances), that they shouldn't worry if they couldn't answer any of the questions, and that they could ask me questions during the interview if there was anything that

they were uncertain about. Also I told them that I wanted them to give their honest opinions.

4.8.3.2 *Difficulties with establishing clear communication*

A further difficulty was in establishing clear communication, particularly in relation to the language which I used to describe the processes of teaching, learning, thinking and problem solving. I was made starkly aware of this difficulty when Paula, in the last five minutes of her fourth interview, asked me what I meant by a strategy, when I had used this term in the written specification of one of the review tasks which we discussed in her first interview, and I had assumed since that point onwards that she understood it. Also there were occasions during interviews when I found that the pupil(s) and I were talking at cross-purposes. One such occasion was when I asked pupils to identify when they had used a particular heuristic strategy for problem solving, 'break the problem down into smaller, more manageable problems', whilst learning to program. I had in mind the top-down design process as the most obvious manifestation of this approach in programming, however some pupils were thinking instead about having applied a problem solving schema – first analyse the problem, then make a plan, implement your plan and finally evaluate the solution, as a way of making the solution process manageable. Even the word 'problem' frequently had different connotations, e.g. the set problem (the one set out within the problem description), and an emergent problem which arises in the course of solving the set problem (such as a program that the pupil has written producing the wrong output). On occasions like this I sometimes failed to respond appropriately to their suggestions because my mind was on a different track, and it was only when I replayed the tape and read the transcription that I realized this. This is something that I struggled with throughout the interviews and when interpreting written comments. Powney (1996) describes the handling of language which is often vague and loosely defined as a recurring challenge of research into values education, and I would suggest that it is also a recurring challenge in any field of research where we are asking children (or adults) to reflect on their own processes of learning and problem solving. Holroyd's (1989) observation on the lack of clear definition of terminology relating to problem solving in Standard Grade course documentation is pertinent here.

4.8.3.3 *Potential omissions, inaccuracies and distortions in the data*

The data gathered from questionnaires, review tasks, interviews, classroom observations and discussions with the class teacher are based on participants'

recollections, which are prone to omissions, inaccuracies and distortions. Other data can be compared with these sources, such as pupils' folders of work, assessment portfolios and progress records. It is not possible, however, in all instances to reconstruct events accurately or fully. As an example, pupils' earlier, unsuccessful attempts at some tasks may not have been retained (e.g. corrections may have been made onto the earlier attempt rather than having been written out separately at the end of the answer booklet) and the end product may provide insufficient clues as to how it was constructed. Therefore I could not always be sure that the things that pupils thought they had done actually happened.

Some studies (e.g. Schoenfeld, 1985) use an experimental setting involving verbal protocols, where individuals/pairs report out loud what they are thinking as they work on problems. However, the setting can place students under considerable pressure, which in turn affects their behaviour. Schoenfeld uses an example to illustrate this, describing students' reactions while solving a novel mathematical problem out loud:

> The cells problem was outside the range of the students' experience, and they found it disturbing. They read the problem statement and were stuck. They had not worked problems like this in a formal setting (if ever), and they had no clear idea how to approach it. Knowing that one or more mathematics professors would be listening to the tapes of their work, the students felt under great pressure to produce something substantial for the record. It goes without saying that the "something" they produced should be mathematical. So the students working alone responded to the pressure by doing the only formal mathematics related to the problem statement they could come up with under the circumstances . . .
>
> (Schoenfeld, 1985, p.280).

He points out that an ostensibly objective evaluation of the students' responses would in this case have resulted in conclusions that didn't make sense, and that this is illustrative of the kind of subtle difficulties inherent in analysis of problem-solving processes. A further disadvantage of using verbal protocols is that, although the technique does yield a very large amount of rich data, the process of gathering and analysing the data tends to be very time consuming and labour intensive (Green, 1995). In the circumstances, I did not feel that it was an appropriate method to use for this study; I did not want to risk putting pupils under pressure, and I also did not want to restrict my interest to only a few pupils working a few problems in an artificial situation.

Schoenfeld (1985) used the example above to make a more general point, that, 'any particular approach to studying intellectual behavior is likely to illuminate some aspects of that behavior, to obscure other aspects of it, and to

distort some beyond recognition. Of necessity the same phenomenon must be investigated with a variety of methodologies, and from a variety of perspectives.' (p.283). Brown (1992) makes a similar point when she highlights the need for those conducting classroom observations to also use probing methods (e.g. open ended questioning), to help the observer to understand events from the perspective of other participants, since the observer's own preconceptions may act as a barrier to understanding.

4.8.3.4 Potential gender bias

The gender balance within the case study class is not typical of the male-dominated pattern within the subject area. The proportion of girls in the class is 60% (the typical proportion of girls taking Standard Grade computing studies nationally is about 35%). Some possible subject-related factors that may influence girls' choice of computing are discussed by HM Inspectors of Schools (1993). Negative influences include an over-emphasis on 'hard' technology at the expense of human issues, lack of opportunity to co-operate and communicate, lack of portrayal of positive female role models in connection with the subject, sexist language and materials, and unequal access to teaching support and resources. Klawe and Leveson (1995) identify a similar set of factors affecting girls' uptake of pre-college courses in the USA[17]. The learning environment within the case study class has been designed to counter these negative influences, in the following ways: (i) There is a focus on human issues through placing the users' needs as a central consideration when designing software. For example, in topic 17 pupils must design a program for primary school children to use, and hence the program must not crash if a wrong key is pressed, the screen layout and presentation must be attractive and uncluttered, and it must be very simple to operate (there should be no need to enter long or complicated commands). (ii) Co-operation and communication are engendered through creating a relaxed atmosphere in the classroom where pupils are encouraged to work with friends, through the use of group interviews and a group review task, and the design of some tasks to require collaboration between pupils. Examples of such tasks are: the gathering of data from the class on a theme of the pupil's choosing to provide the input data for a program which outputs tables of information sorted into different orders (in topic 7); testing out each other's quiz programs, where the quiz theme is

---

17 There are, of course, influences beyond these, such as in the family, society, the education system and industry. Klawe and Leveson (1995) quote the percentage of women gaining Ph.D's in computer science as remaining steady at between 12% and 13% since 1980, in contrast to all other fields of science, maths and engineering, where there has been a steady upward trend.

selected by the pupil (topic 15); and learning how to read music by having it explained by another pupil in the class who can already read music, in order to encode the information on each musical note to get the computer to play a tune (topic 17). (iii) The class teacher and I should present positive role models of women in computing. There is a balanced portrayal of male and female roles within the learning materials with a deliberate choice of contexts that are gender neutral, in marked contrast with some other resources which are still in common use in Scottish schools to teach non-programming aspects of the Standard Grade course, such as a database study unit which is based entirely on processing information on football clubs, and an SEB project on football league tables (SEB, 1993a). (iv) The class teacher will establish with pupils a strict regime to ensure that the arrangements for getting time on the computer are equitable, and this will be monitored by including an item on access to computers in the unit 1 questionnaire. Formative assessment procedures will ensure that *every* pupils' work is checked regularly, so that all pupils should enjoy equal access to teaching support.

4.8.4 An examination of my role in the research, and whether the study could be replicated by others

My role in the case study research does not fit the conventional mould of the 'insider researcher' (the teacher researching into his or her own practice) or the 'outsider researcher' (the researcher approaching a situation in which he/she has had no previous involvement). To the pupils I was viewed as an outsider (the class teacher recounted to me their question one day: 'Is that wumman comin' today, miss?'), someone who got them talking and writing about their work, and who knew about programming and so could help them if they were stuck. To the class teacher, I was a teaching partner (and, occasionally, I suspect, a nuisance!) who shared her goal of maximizing the quality of pupils' learning, and also a fellow researcher (since she was able to perceive her role in this way). To other teachers in the school, I was a familiar face in the staff room, having 'shadowed' two classes over the previous session as part of the process of field testing the learning resources.

That my role should combine a teaching and research function is appropriate, in order to fit into a reflective professional culture:

> Within a reflective professional culture 'teacher' and 'researcher' are two aspects of a single role in which teaching constitutes a form of research and research constitutes a form of teaching. (Elliott, 1991, p.64)

Stenhouse (1983, p.192) argues that researchers have an obligation to ensure that their interventions benefit pupils' learning, describing action research as:

> a pattern of research in which experimental or research acts cannot be exempted from the demand for justification by professional, as well as by research, criteria. The teacher cannot learn by enquiry without undertaking that the pupils learn too ...

Was my role as a researcher made any easier or more difficult as a consequence of being partly 'on the inside'? I believe it was made much easier by being already conversant with the learning resources and their use, and I could therefore concentrate on getting to know the pupils and their programming work. This did not diminish my concern to subject the resources to a critical scrutiny, since it was through pupils' engagement with the resources that they would be appraised. The understanding and trust that the class teacher and I had built up through the programming project ensured that we worked together in harmony.

Was I less objective because I felt a sense of ownership of the developments? This question is never easy to answer, other than to observe that the vision of, 'a dispassionate, objective view of events by an unbiased observer,' (Brown, 1992, p.1) is never realized in practice because every observer brings biases and prejudices to their work. Brown's advice was heeded, on making explicit my underlying assumptions and theories so that biases can be taken into account, and probing the perspectives of others to understand better what is going on.

Could the learning and research environment be successfully re-created by the class teacher if I was not present, or by other project teachers with their classes? I believe that it could, if my role was taken on by another teacher in the department team-teaching for a proportion (at least a third) of the time while pupils are learning programming, or if the procedures were adapted to enable a teacher to conduct the class on his or her own. It would not be necessary or desirable to attempt to reproduce the learning and research environment precisely as the learning needs of each new class must be appraised afresh.

Could the learning and research environment be successfully re-created by teachers who have not participated directly in the programming project but who do have access to the learning resources? Again, I believe that it could, if the principles and methods were fully explicated, and if Harlen's (1994) advice on the timescale of evaluations is heeded (principally that teachers need time to familiarize themselves with new ideas and approaches before it is reasonable to expect positive effects).

Progress Chart

	S3 Term 1 Sep - Dec 1993	S3 Term 2 Jan - Mar 1994	S3 Term 3 Apr-Jun 1994	S4 Term 1 Aug-Oct 1994	Total absences
Bryan	----------x--x-----x--1----xx--	----------x	-----------2---------	----------xxx----------	9
Stephen	-x---x----------x----------1-	----x------	-x---x-----x--xxxxxxx	---x-----xxxx---------x	20
Claire	------------------x--1--------	----------	---------2--x-------x	--------------x-------	4
Scott	----------x----------x1-xx----	----------	-----------2--x------	----------	5
Paula	----------------1x---------xx--	--------2--	----------	-----3----------	3
Claudia	------------x------1xx------x--	--x--x-----	-----2------x-----xxx	---------------x------x	12
Carol-Anne	------x---------xx---1-x--xx---	--x---x----	---------2---x----xxx	----xxx--------xx-----x	18
Dawn	------------x------1----------	----x------	-x-x--x-----2-----x--	----------x----------	7
Chris	------------1-x---------------	----2------	-x----------x-x------	-----3----------	4
Caroline	--------------------1----------	----------	-----2----xx---------	------------------3----	2
Anthony	xx---------------x-xxx----1----	-----x-----	--xx----------x---xxx	----------2---------x--	14
Michael	----------x--x-x-1-xxx---xxxxx	-----x-x--	----------x----x	---------------------x-	16
Kirsty	----------------xx--1----------	-----x-----	------2-------------x	--------------3--------	4
Bernadette	--------------------1-x-x---x--	-x------xx	-------2-x---x-------	----------	8
Kelly	------------xxx-------1-------x	-----x-----	------2------------xx	----------x---------x3-	9
Catherine	-------------xx--x--1----x-----	---xxx----x	-x-----2-----x-----xx	----------x---x--x---x	16
Kathleen	------------------1------------	----------	-----2-------x-------	-------xx-------------	3
David	----------------------x1--------	----------	------x------x-2----x	----------	4
Gerard	-------------x--------x--x1----	----xx-----	---------------xxxxxx	----2---x-------------	12
Lorraine	----x------x---1----xx---------	-----2-----	-------------x-x---xx	--------3------xxx-----	11
	[31 periods]	[11 periods]	[21 periods]	[23 periods]	

Key:

-	each period spent on programming
x	absence
1 (or 2 or 3)	when the pupil finished unit 1 (2 or 3).

Figure 5: Progress of individual pupils and absences.

5 CASE STUDY FINDINGS I: PROGRESS

5.1 CHAPTER OVERVIEW

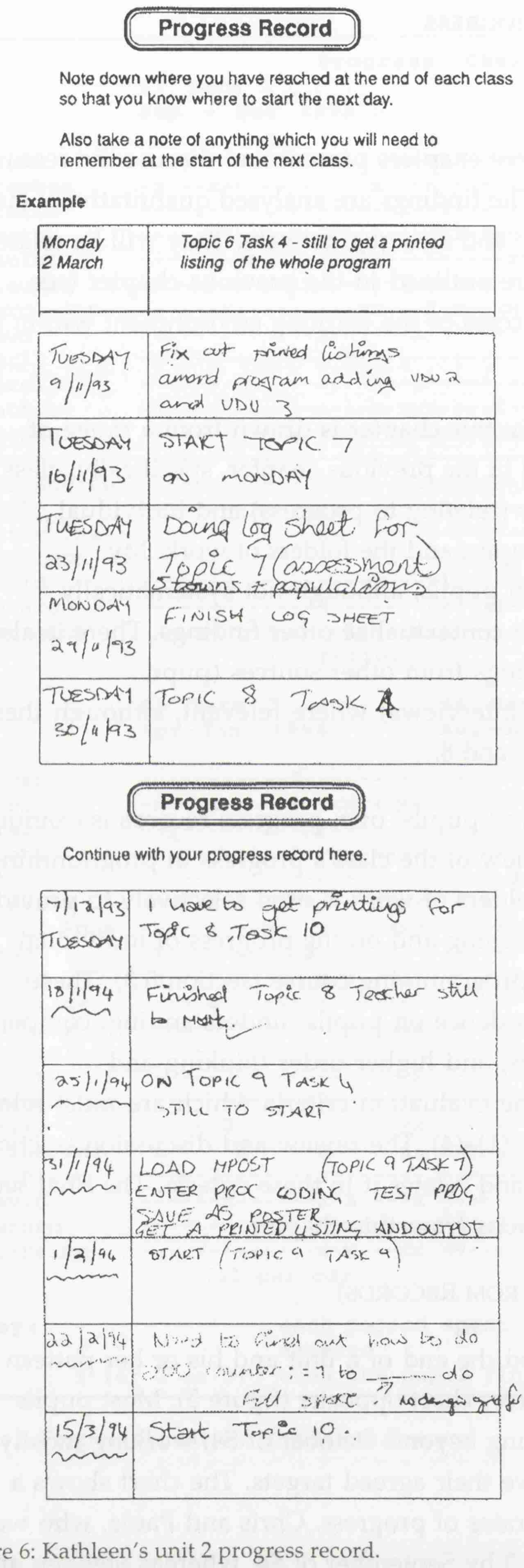
This chapter and the following three chapters present and discuss the research findings for the case study class. The findings are analysed quantitatively (as frequency counts and percentages) and also qualitatively. They will be related to the evaluation criteria which are outlined in the previous chapter (see figure 4) in order to gauge the success of the learning environment within the case study class.

The evidence which is presented in this chapter is drawn from a range of sources, each of which I described in the previous chapter, specifically: class records maintained by the teacher (relating to progress) and individual progress records maintained by pupils; and the folders of work. My observations and interactions with pupils, although not systematically recorded, are used on occasions to contextualize other findings. There is also some occasional reference to findings from other sources (pupil questionnaires, review tasks and interviews) where relevant, although these are mainly discussed in chapters 7 and 8.

The evidence from class records and pupils' own progress records is examined first in order to provide an overview of the class's progress in programming (section 5.2). Evidence from the folders of work is used selectively to provide a detailed focus on teaching and learning and on the progress of individual pupils at an interim stage of the programming course (section 5.3). These sources together provide some evidence on pupils' understanding, competence at solving programming problems, and higher order thinking and metacognitive skills. Therefore the evaluation criteria which are most relevant to the findings in this chapter are (1)–(4). The review and discussion (section 5.4) draws the evidence together and relates it to these criteria. The final section (section 5.5) presents the conclusions from this chapter.

5.2 EVIDENCE ON PROGRESS (FROM RECORDS)

The time when each pupil reached the end of a unit and his or her pattern of absence are indicated in the progress chart opposite (figure 5). Most pupils continued to work on programming beyond October of S4, working mostly in their own time, in order to achieve their agreed targets. The chart shows a considerable variation in pupils' rates of progress. Chris and Paula, who were furthest ahead, had finished unit 3 by September of S4, whereas Stephen and



Progress Record

Note down where you have reached at the end of each class so that you know where to start the next day.

Also take a note of anything which you will need to remember at the start of the next class.

Example

Monday 2 March	*Topic 6 Task 4 - still to get a printed listing of the whole program*

TUESDAY 9/11/93	Fix out printed listings amend program adding VDU 2 and VDU 3
TUESDAY 16/11/93	START TOPIC 7 on MONDAY
TUESDAY 23/11/93	Doing log sheet for Topic 7 (assessment) 5 towns + populations
MONDAY 29/11/93	FINISH LOG SHEET
TUESDAY 30/11/93	TOPIC 8 TASK 4

Progress Record

Continue with your progress record here.

7/12/93 TUESDAY	I have to get printings for Topic 8, Task 10
18/1/94	Finished Topic 8 Teacher still to mark
25/1/94	ON TOPIC 9 TASK 4 STILL TO START
31/1/94	LOAD MPOST (TOPIC 9 TASK 7) ENTER PROC CODING TEST PROG SAVE AS POSTER GET A PRINTED LISTING AND OUTPUT
1/2/94	START (TOPIC 9 TASK 9)
22/2/94	Need to find out how to do 200 counter: = 1 to ___ do Fill space no change given for Topic 8 Task 11
15/3/94	Start Topic 10.

Figure 6: Kathleen's unit 2 progress record.

Michael were still working on unit 2 in October of S4. For some pupils (such as Stephen, Carol-Anne, Michael and Catherine) the frequency and pattern of their absences may have slowed their progress considerably. Not only would they have less time in class to do programming than the others, but it might take them some time to pick up the threads of their previous work whenever they returned following a period of absence. Michael, for example, had made very good progress at the beginning of the course; he was actually furthest ahead for the first five weeks. Then, following numerous absences and a brief suspension from school, he fell behind. He took from the end of October till mid-January of S3 to do topic 6, by which time Chris had finished topic 9. He seemed to lose interest in the work.

The progress record (at the front of the answer booklet for each unit) was designed to help pupils monitor their progress from day-to-day and start each programming lesson smoothly at the point where they had left off the previous day. Not all pupils maintained it regularly, and some did not include enough details for it to be very useful. Taking, for example, the unit 2 progress record for the four pupils with the highest frequency of absence, Michael's has only four short entries, one of which is a reminder from the class teacher to Michael that his assessment task is incomplete; Carol-Anne's has large gaps and contains only the topic and task number that she was working on; Stephen's is completely blank, and only Catherine's has been filled in regularly, albeit that the entries are brief. This contrasts with some other pupils' progress records, such as Kathleen's (figure 6). The entries in Kathleen's record are varied and detailed, e.g. 'Fix out printed listings amend program adding VDU 2 and VDU 3', 'I have to get printings for...', 'Finished topic 8. Teacher still to mark.', 'Need to find out how to do...'. From this it can be seen that Kathleen is very actively engaged in managing her learning. It would appear that Stephen, Carol-Anne and Michael in particular are doubly handicapped by having a high absence rate and by not maintaining an adequate record of their progress.

Figure 7 (see over) provides further information about progress by indicating the last topic that each pupil in the class had finished by the end of March of S4 when coursework grades were due to be submitted to the SEB. It also shows the final progress target that each pupil had agreed with the class teacher. From this it can be seen that most pupils achieved the target and one pupil (Anthony) exceeded his. The girls made better progress than the boys with an average difference of 1.5 in the number of topics completed.[18]

---

18 This figure is given as a rough indicator only, since one should not compute an average unless the points are on an equal interval scale.

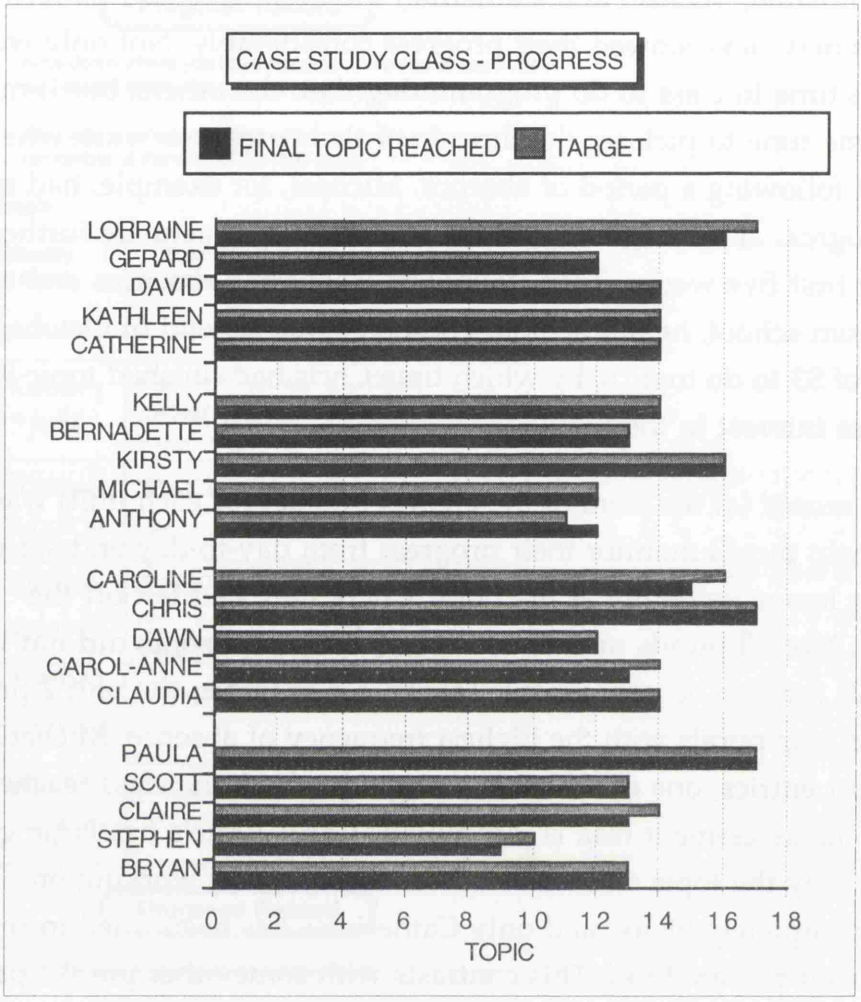


Figure 7: Pupils' progress by the end of S4, and their agreed targets.

Topic	1	2	3	4	5	6	7	8	9	10
Average Time (In Periods)	1.2	1.8	5.6	5.2	5.9	5.2	4.3	5.7	6.4	6.1
Minimum Time (In Periods)	1	1	4	3	3	2	3	2	4	2
Maximum Time (In Periods)	2	3	8	8	10	11	11	11	12	9

Table 7: Average, minimum and maximum times (given in periods of 52 minutes) to complete topics.

On the surface, the class's progress seems disappointing when one considers that the recommended time for the programming topic is 40 hours and the actual time spent in class was approximately 70 hours. Even with some pupils doing extra work in their own time, no-one managed to finish all four units. That this has happened is not altogether unexpected given the findings from other studies which indicate that it generally takes learners a considerable time to learn to program well and to become sufficiently competent to use the more advanced features of the language they are learning. Cope and Walsh (1990) cite a number of studies in which learners did not, in the opinions of the researchers, make much significant progress when the time that they had spent on learning programming was short (e.g. 90 minutes per day for six weeks, which is 45 hours), and in Kafai's study (1993) pupils had to be given extra time to complete their projects.

When one examines the time that it took pupils to complete individual topics (measured by the number of periods and excluding absences) wide variations between pupils emerge. Table 7 (opposite) shows the average, minimum and maximum time spent on units 1 and 2 topics. Why does it take some pupils five or six times longer than others in some topics, to do the same work? Perhaps part of the explanation lies in Papert's observation on the tendency of some pupils to start over and over again whenever they encounter problems with a solution (see 2.6.1), in addition to the factors discussed at the beginning of this section?

5.3 EVIDENCE FROM THE FOLDERS OF WORK

5.3.1 Method of analysis

I shall focus here upon pupils' work in topics 9 and 10. My choice of these topics was guided by four factors. Firstly, the topics are linked, with topic 10 extending the themes of topic 9, and taken together they represent a reasonable episode of learning (the time that pupils spent on both topics ranged from 7 periods to 20 periods). Secondly, all pupils completed topic 9, and only Stephen did not complete topic 10, so that almost everyone's work can be examined. Thirdly, the topics encompass most of the key aspects of the content which are taught in units 1 and 2 and there is therefore a good opportunity to assess pupils' competence and understanding across a range of aspects. Fourthly, by this stage of the programming course the complexity of the problems is beginning to build, and it is likely that some, perhaps most, pupils will experience some difficulties with the work. An analysis of the nature and extent of their difficulties will allow an assessment to be made of pupils'

```
100 // Text input and output, Topic 9, Task 2
110 // [pupil inserts name here]
120 // [pupil inserts date here]
130 MODE:=7
140 // tell the user what the program does
150 PRINT "This program prompts you to enter a"
160 PRINT "name and then displays it 10 times."
170 PRINT
180 // prompt for, and input a name
190 PRINT "Please enter a name."
200 INPUT name$
210 // output the name ten times
220 PRINT
225 FOR counter:= 1 TO 10 DO
230   PRINT "The name that you entered was "; name$
235 NEXT counter
240 END

]RUN
This program prompts you to enter a
name and then displays it 10 times.

Please enter a name.
?Andy

The name that you entered was Andy
[shown ten times]

DESIGN
1    select mode 7
2    tell the user what the program does
3    prompt for, and input a name
4    leave a blank line
5    loop ten times
6       output the name
7    end loop
```

Figure 8: Topic 9, first problem on input and output of text (unit 2 task sheets p.36–8).

developing understanding and competence as they progress through the problems in topics 9 and 10, and of how effectively they manage their learning.

5.3.2 *An overview of topics 9 and 10*

These topics extend pupils' knowledge and understanding of language features, programming concepts and principles, standard algorithms, and problem solving processes. Pupils are introduced to using string variables (they learned about numeric variables in the previous topic), and to using a FOR loop to repeat a sequence of commands. Three of the four programs involve the use of procedures, which pupils first learned to use in topic 5 (in topic 5 they wrote their own procedures to output text, and in topics 6 and 7 they used pre-written procedures to input, sort and output text or numbers). All of the programs in topics 9 and 10 utilize the same standard algorithm – *input, then loop to output*[19]. Knowledge and understanding of design is extended by demonstrating how the main steps of a design can be further sub-divided into smaller steps until the design has been worked out in sufficient detail to implement it (a top-down approach). Knowledge of testing and debugging strategies is extended through providing explicit guidance on the nature of the input data that are required to test the programs systematically, and through a debugging task in which pupils must describe how they identified and rectified each bug in a program, leading (in the summary sheets) to a consideration of effective testing and debugging strategies.

The first problem introduces the input and output of text (the program prompts for a name to be entered and then displays it), and examines how to adapt the program to output the text a fixed number of times, using a FOR loop. This is a very straightforward problem which none of the pupils encountered any difficulties with. Pupils ran and tested the programs several times, examined and compared the listings, edited the second version to output the name 20 times instead of 10 times, and examined the design for the second version to see how one can indicate that certain steps have to be repeated (figure 8).

The next three problems utilize the same algorithm – *input, then loop to output* – however the problems are more complex, involving the use of procedures, more than one input, and formatting of output to the screen and to the printer. In the last problem the value of the loop counter is also output

---

19 In Soloway's terms (1986), this would be described as a plan. Linn and Clancy (1992) would describe this as a program component around which template knowledge is constructed.

Problem Description

Each 3rd and 4th year class is running a fund raising event to be held one afternoon.

You have to design and write a program to print posters, like this:

```
============================
Talent competition
Assembly hall
Tickets 20p
============================
or
============================
Staff v pupils football match
Playing fields
Tickets 10p
============================
```

The program should prompt the user to enter the details which are to appear on the poster.

The output should consist of three printed copies of the poster.

Figure 9: Topic 9, the problem description for 'posters' (unit 2 task sheets p.39).

each time that the loop is obeyed. However, beyond the initial problem description, each solution unfolds differently. The three problems are referred to below as 'posters', 'address cards' and 'prize-draw'.

5.3.3 *The 'posters' problem*

The problem description for 'posters' is shown opposite (figure 9). The solution is obtained in stages beginning with analysis of the problem and design of the solution, followed by implementation and evaluation. These stages can be roughly equated with Polya's (1948) advice on how to solve a problem; first understand the problem, then devise a plan, carry out the plan, and look back. The materials present a model of the solution process which is reinforced by the teacher's pedagogical interactions with pupils. The approach differs from the common device of using a 'worked example', a feature of many programming and mathematical texts (e.g. Koffman, 1985) since at each given stage the pupil is required to contribute part, and sometimes all, of the solution. In this way, pupils must always integrate their existing knowledge of programming techniques and problem solving strategies within a new framework.

From the initial statement of the problem, the solution unfolds as follows:

Task 4 Apart from the type of event, what other details will the user have to enter? Also show what your poster will look like. (Analysis and design.)

Task 5 Here is part of the design for the program. Notice that step 3, 'prompt for, and input the poster details', and step 6, 'output the poster', have been partially broken down into smaller, more detailed steps. Complete the refinements for these steps. (Further analysis and design.)

Task 6 Since steps 3 and 6 have been broken down into smaller steps, you can use a procedure for each in the program. Given part of the program (the main program and the first few lines of each procedure), finish coding it. (Implementation.)

Task 7 Enter the program (you can load the main program to speed up the process), and test it by inputting the details that you have shown on your poster design. Obtain a printed listing and printed output. (Further implementation and evaluation.)

Task 4

1 The other details are where it is to be held and the price of the tickets.

2

```
==============================
Concert
Theatre

Tickets 50p
==============================
```

Task 5

```
Design for the poster program

1 select mode 7
2 tell the user what the program does
3 prompt for, and input the poster details
4 start printing
5 loop three times
6   output the poster
7   leave a blank line
8 end loop
9 stop printing


Step 3
1 prompt for, and input the type of event
2 prompt for, and input the place
3 promp per, and input the price


Step 6
1  output border (like this,  ======  )
2  output type of event
3 output a blank line
4 output the place
5 output a blank line
6 output the price
7 output border (like this, ======)
```

Figure 10: Catherine's design for 'posters', topic 9 tasks 4 and 5.

Task 8 Make up a poster with different details and obtain two printed copies of it. (Further evaluation.)

Task 9 Edit the program to let the user choose the number of posters that he/she wants. (Enhancement of the solution.) Do further testing (two test runs) and obtain a printed listing. (Evaluate again.)

5.3.4 Pupils' progress with the 'posters' problem

It must be borne in mind when examining the written evidence that it may not represent pupils' first attempts at the solution; some pupils may have written out a rough version of their answers on a separate sheet before copying them into their answer booklets, and others may have made corrections onto the answer booklet. Also any printouts relate to the final versions of programs. Hence pupils may have encountered more difficulties than is apparent from scrutinizing the evidence in their folders of work.

Very few errors were evident in pupils' work from tasks 4 and 5. Pupils seem to have understood, from the description of the problem, the nature of the inputs to the program and the nature of the output that it would produce, and they also seem to have understood what they were being asked to do in relation to refining two of the steps. Catherine's design for 'posters' (figure 10) illustrates this.

However in task 6, where pupils had to complete the coding for each procedure, and in task 7, where they had to enter and test the program, there is evidence from the written responses (on p.12 of the answer booklet) and printouts that most pupils encountered some difficulties. There is a distinct pattern of errors (table 8 overleaf); any errors which could be detected have been included in the analysis, even if they were subsequently corrected by pupils. The analysis extends also to the printed listing which pupils obtained for task 9. The last column indicates how many periods each pupil took to complete the topic overall.

There is some evidence missing from pupils' folders. A few pupils failed to retain printouts, in spite of reminders from the class teacher, but it was possible to retrieve all but two printed listings by accessing pupils' user areas (the computers are networked and so the class teacher has access to all of the files centrally). When Gerard was working on topic 10 he took his answer booklet home to do extra work and failed to return it, and therefore all of his written work for unit 2 is now missing.

'Posters'	Written coding (Task 6) X = error					Printed listing (Task 7)	Printed listing (Task 9)	Time on topic (periods)
	E1	E2	E3	E4	E5			
Bryan					X	Correct	Same error (E5) as in task 6. [1]	4
Stephen						Correct	Correct [1]	8
Claire		X			X	Correct	Correct	10
Scott	X	X			X	Correct	Correct	5
Paula						Correct	Correct	4
Claudia	X				X	Correct [1]	Correct	6
Carol-Anne	X	X			X	Correct	Missing [2]	10
Dawn	X			X		Correct	Correct	11
Chris	X					Correct	Correct	5
Caroline		X		X	X	Correct	Correct	6
Anthony	X					Correct	Correct	12
Michael		X	X	X		Correct	Correct	5
Kirsty	X	X				Correct	Design, program mismatch.	5
Bernadette	X	X				Correct [1]	Missing prompt.	4
Kelly						Correct	Correct	4
Catherine	X	X			X	Same error (E5) as in task 6.	Correct	4
Kathleen	X		X			Correct	Correct	7
David		X			X	No output to printer.[1]	Design, program mismatch.[1]	8
Gerard	Answer booklet missing					Correct	Missing [2]	5
Lorraine	X		X			Correct	Correct	5
Frequency of errors	**11**	**9**	**3**	**3**	**8**			

Key:	Written coding
E1	Missing $ (numeric instead of string variable)
E2	Missing END PROC
E3	'OUTPUT' instead of 'PRINT'
E4	Variable or procedure name mismatch
E5	Output of a constant instead of a variable
	Printed listing
1	Not in folder, retrieved from pupil's user area.
2	Not in folder and not saved, therefore missing.

Table 8: Pupils' progress on the 'posters' problem of Topic 9 (unit 2 task sheets p.39–44).

In task 6 only three pupils (Stephen, Paula and Kelly) produced error-free solutions. The most frequent types of error when completing the coding for each procedure were using a numeric instead of a string variable, omitting the 'END PROC' command, and outputting a constant instead of the value of a variable. Catherine's written coding (figure 11 overleaf) includes all three of these error types.

As pupils entered their programs (in task 7), some new errors were introduced, for example, if a pupil typed in '1080 END PROC inpu_details' a name mismatch error would be indicated when running the program because the procedure name had been entered incorrectly. On the other hand, some errors in the written version of the program were noticed and corrected by pupils as they entered programs, for example, one of the error types (E3) would generate an error message ('Syntax error at line ...'). Three of the other error types (E1, E2 and E4) would generate an error message when running the program (e.g. 'Bad value' if a string is entered by the user when a numeric is expected in the program, or 'Unclosed at...' if the END PROC command has been omitted, or 'Not found at line...' if a wrong variable name has been used). Three of the error types (E1, E2 and E4) were covered in the help hints for topic 9 or the help hints for earlier topics, although not all pupils were in the habit of using them regularly (this is a finding from the unit 1 questionnaire).

Once pupils had tested and debugged the program in tasks 7 and 8, only Catherine's program failed to meet the specification; it is discussed below. Since the main program was given and pupils had only to complete the coding for each procedure, this reduced the scope for pupils to introduce logical errors, and it therefore reduced the degree of difficulty that pupils might have experienced with debugging. Nevertheless, it is clear from comments during interviews and from written responses to the first review task that some pupils did find it difficult to debug the program and they spent a considerable amount of time on debugging. For example, this is part of Caroline's response to the first review task, in which she refers to her work on the 'posters' problem:

> *...I enjoyed working on this program mainly because it required a lot of thinking and testing. It took me a bit longer than some other programs I have previously written but was a lot more interesting. Every time I ran the program I would discover another bug but eventually I managed to correct them all and get the program working. I learned that having a detailed design to work from really does help, and I think that had I not used a design I would have had a lot more errors.*

The last problem in topic 10, 'prize-draw', will present an opportunity to examine pupils' testing and debugging skills in more depth.

```
1000 PROC input_details
1010   // prompt for, and input the poster details
1020   PRINT "Please enter the type of event"
1030   INPUT type_of_event$

1040   Print "Please enter the place"
1050   INPUT the_place.
1060   Print "Please enter the price"
1070   INPUT the_price.


2000 PROC output_poster
2010   // output the poster
2020   PRINT "============================="

2030   Print "Concert"
2040   Print
2050   Print "Theatre"
2060   Print
2070   Print "50p"
2080   Print "============================="
```

[Catherine's written additions to the program are in italics.]

Figure 11: Catherine's written coding for 'posters', topic 9 task 6.

In task 9, when making the final modification to the program, both Kirsty and David implemented it by placing the prompt and input for the number of posters within the 'input_details' procedure rather than within the main program, which did not affect the running of the program but it meant that the design and program did not correspond. Bryan has introduced an error in task 9 by outputting constant values (using PRINT and placing the text within quotes, in the same way that Catherine did – see figure 11) within his 'output_poster' procedure. The only other error was a missing prompt in Bernadette's program.

Having presented an overview of pupils' performances in the 'posters' problem, I shall now return to examine each error type (E1–5) in order to explain why this pattern of errors may have occurred at this stage of their learning, using appropriate illustrations.

5.3.4.1 Error 1 – using the wrong variable type (numeric instead of string)

This is the error that Catherine made at lines 1050 and 1070 of her 'input_details' procedure (see figure 11). This type of error seemed to cause the greatest difficulty for pupils, on the basis of the frequency of its occurrence, its recurrence in the next solution (although there were fewer instances), pupils' assessments of their confidence about this aspect in the unit 2 questionnaire, and from comments made by pupils during interviews. This is in spite of the hint provided in task 6, where lines 1020 and 1030,

```
1020 PRINT "Please enter the type of event"
1030 INPUT type_of_event$
```

show how to implement the first refinement for step 3 of the design, 'prompt for, and input the type of event'.

All pupils are familiar with using variables which represent numbers from mathematics, as well as having recently encountered numeric variables in a programming context in topic 8. The types of problems in topic 8 (one of which is about calculating an area and another calculating the average of two numbers) would help to strengthen the association with mathematics. However the idea of using a variable to represent text is unfamiliar, except perhaps for those few pupils who have done some programming before. Also the text may actually be or contain a number, such as a telephone number or postcode; the important point is how you intend to process it in the program

(e.g. you wouldn't normally wish to add a list of telephone numbers together), and this is quite a subtle distinction.[20]

Later, in task 9, pupils are asked to identify the type of value (numeric or string) that should be input by the user for the number of posters required, and to choose a suitable variable name to represent this value. It is clear that a number of pupils had difficulty with this, as there is lots of scoring out and some incorrect answers. Kathleen and Catherine, for example, had said 'string' and written down 'poster$' as their choice of variable name, whereas Chris had also said 'string', but his choice of variable, 'number_of_posters', was a numeric type. David and Michael had correctly identified the type of variable as numeric, but they had chosen 'poster$' as their variable name (although Michael had subsequently scored out the $ symbol). Stephen had changed his mind twice about the type of variable before eventually getting it right. This pattern of errors indicates that for some pupils it isn't just a problem with syntax (knowing when and when not to use the $ symbol), or forgetting to include the $ symbol, but rather with understanding the distinction between the two variable types. However none of their finished programs included this error, and clearly those pupils who were having difficulty had either sought out help or they had managed to detect and correct the error for themselves. Dawn and Bernadette were the only ones to choose to represent the entrance cost for their event as a numeric variable. (Dawn's program included this (correct) line: `PRINT "TICKETS "; price_of_tickets; "p"`)

Pupils' knowledge of using string variables and the distinction between the two variable types is inevitably fragile at this stage of their learning since the ideas have been introduced for the first time in this topic. The 'posters' problem is a good vehicle because it serves to reveal the extent of pupils' fragile knowledge.

5.3.4.2 *Error 2 – failing to include the 'END PROC' command*

This is the error that Catherine made at lines 1080 and 2090 of her 'input_details' and 'output_poster' procedures (see figure 11).

Some considerable time has elapsed since pupils last wrote a procedure (in topic 5), although they have been working with pre-written procedures in topics 6 and 7. Given the large amount of details about language constructs that pupils have to learn to produce even simple programs (a printout of on-screen

---

20 There is an example given within Topic 9 of how one cannot use a string variable (cost$) as part of a calculation to work out a discount (Unit 2 task sheets, p.35).

information on the various commands introduced in unit 1 fills nine sides of A4[21]), it is not surprising that some aspects are forgotten about over time or carelessly omitted. This is an instance of missing (i.e. forgotten), or, more likely, inert knowledge.

5.3.4.3 Error 3 – using 'OUTPUT' instead of 'PRINT' to output the value of a variable

Some pupils included a line such as '2030 OUTPUT type_of_event$' within the 'output_poster' procedure.

The word 'output' appears frequently in the design for the program, as in 'output type of event', however the corresponding command in the program is not 'OUTPUT' but 'PRINT'. The word 'input' also appears in the design, as in, 'prompt for, and input ...', however it *is* used as the corresponding command in the program. This seems likely to cause confusion. This type of confusion is difficult to avoid entirely since the language that one uses to describe, in English, the steps that the program must go through and the effects that it produces, is bound to overlap in places with the commands available in the programming language, particularly if the programming language has been designed to be readable. Spohrer and Soloway (1986) classify this type of error as construct based and arising from confusion with natural language.

Another possible explanation is that some pupils may not have grasped that the design is written in English as a representation of the solution, and that it cannot be entered directly to the computer. This does not seem to be a very likely explanation in this case, since otherwise one would expect to find other similar errors appearing in the written version of the coding. Much earlier in the course when pupils were attempting the first coursework assessment in topic 3, some pupils had confused their list of steps (the design) with the program itself, however following feedback from the class teacher this problem did not recur in any of their later work.

Perhaps, as Perkins and Martin (1986) suggest, this instance of misplaced knowledge has arisen because, '...the programmer lacks the general critical sense that one simply cannot expect to throw things together in a programming language and have them work.' (p.223) This critical sense may not be fully developed in all pupils, since otherwise one would expect that

---

21 This resource was not available to the case study class because it had been produced after Claudia had suggested in her unit 1 interview that such a resource would have been of help to her and to other pupils.

```
 100 // Program to output 3 printed posters
 110 //
 120 //
 130 MODE :=7
 140 // tell the user what the program does
 150 PRINT "This program prompts you to enter the"
 160 PRINT "poster details and then prints 3 copies."
 170 PRINT
 180 input_details
 190 VDU 2 // start printing
 200 FOR counter:= 1 TO 3 DO
 210   output_poster
 220   PRINT
 230 NEXT counter
 240 VDU 3 // stop printing
 250 END
1000 PROC input_details
1010   // prompt for, and input the poster details
1020   PRINT "Please enter the type of event"
1030   INPUT type_of_event$
1040   PRINT "Please enter the place"
1050   INPUT the_place$
1060   PRINT "Please enter the price"
1070   INPUT the_price$
1080 END PROC input_details
2000 PROC output_poster
2010   // output the poster
2020   PRINT "============================="
2030   PRINT "CONCERT"
2040   PRINT
2050   PRINT "THEATRE"
2060   PRINT
2070   PRINT "50P"
2080   PRINT "============================="
2090 END PROC output_poster
```

Figure 12: Catherine's printed listing for 'posters', topic 9 task 7.

their efforts to produce a written version of the coding would be more painstaking.

5.3.4.4 *Error 4 – changing the names of variables and procedures part-way through the program*

Whenever pupils became aware that they had done this (whether they were able to discover it for themselves or they needed an explanation of why they were getting an error message when they ran the program), it was clear that it was a casual error because they were always able to correct it without further explanation from the teacher.

Pupils in this class were far more likely to have to leave a program part-finished and continue it the next day or whenever they could next get access to a computer during the lesson, than, for example, undergraduate students working in a lab where they would enjoy undisturbed access to computers for longer periods. In circumstances where pupils' concentration is often disrupted one might expect more casual errors to arise in their work, unless they guard against this happening by checking constantly, which not all pupils do (whether it be in the programming class, or in any other class).

Also it may be difficult for inexperienced programmers to know what types of errors to look out for, so that they cannot check through their work very efficiently.

5.3.4.5 *Error 5 – outputting a constant instead of the value of a variable*

This error was more difficult to detect, since it did not generate any error messages when the program was entered or run. It arises in the 'output_poster' procedure, and is illustrated by Catherine's program (figure 12) – note that this shows a listing of her finished program in which all errors in the written version of the program have been corrected except this one. Instead of outputting the value of a variable each time, using 'PRINT' followed by the appropriate variable name (e.g. PRINT type_of_event$), she was outputting a constant value (see lines 2030, 2050 and 2070).

When she tested the program, she entered as data the same information that she had included in each PRINT statement[22]. While she should have detected this bug in her program in task 8, where different input values are required,

---

22 Another pupil, Scott, had made a slightly different error, in which he placed the variable name inside quotes, i.e. PRINT "type_of_event$". This is a much easier bug to detect because the output which the program produces is unexpected and obviously incorrect – the variable name itself is displayed.

```
]run
This program prompts you to enter the
poster details and then prints 3 copies.

Please enter the type of event
?L
Please enter the place
?;
Please enter the price
?K
==============================
BADMINTON TOURNAMENT

P.E. LARGE GYM

30P
==============================
[etc.]
```

Figure 13: Catherine's dummy run of the program from task 7 of topic 9.

she had instead edited her 'output_poster' procedure to change the text inside the quotes (from PRINT "CONCERT" to PRINT "BADMINTON TOURNAMENT" etc.). She had actually tested her program on several occasions using dummy data before obtaining a sample run for her folder (figure 13), and she had either failed to notice that the details on the poster differed from the values she had entered, or she did not appreciate that the two ought to have matched (i.e. she does not understand the strategy of testing a program using dummy data). Unfortunately Catherine's buggy program was only detected when her work was being checked over at the end of the topic, although she had managed to correct the mistake in her version of the program for task 9.

The manner in which the class teacher or I intervened if we spotted this error on anyone's screen listing – after having stood back for a while to provide an opportunity for the pupil to detect the error on his or her own – was to suggest that the pupil run the program again and type in different details (e.g. 'What about a quiz?'). Of course the details which would then appear on the poster would not match the inputs, alerting most pupils to the problem. The teacher and I were therefore *modelling* the correct approach to adopt when testing a program, i.e. you should always predict the output that you expect to get, and test with different inputs. This intervention provided a natural opportunity to discuss correct procedures for testing the program with the pupil. The 'posters' problem is a good problem vehicle because it offers such opportunities.

What other reasons are there for this error occurring? Some pupils seemed to become fixed on their poster idea and found it to be very appealing (e.g. Claire had in mind a, 'Throw wet sponges at teachers competition', and David, in common with most of the boys, a Celtic football match). Because of this they seemed to lose sight of the fact that the program should be able to produce posters which contain different information, depending on what the user wants. Perhaps the concept of the 'user' is problematic when pupils' programs are in reality mostly only ever used by the person who has created them; the lack of authenticity makes it harder for pupils to design programs with the needs of a real user in mind. While the problem description was framed in such a manner as to suggest a potential user (pupils in another 3rd or 4th year class), this suggestion may have been forgotten about, or deliberately discarded by some pupils as irrelevant or uninteresting. Everyone in the class has an important role to play by acting as potential users, to remind pupils that they are engaged in creating a *product* which others will wish to use, and which has

to satisfy certain requirements[23]. DiSessa's concept of the computer literate classroom which operates through sharing ideas and artefacts is pertinent here (see 2.7.1). Soloway (1986) advocates that students should be taught explicitly that programming is a design discipline, in which the output of the programming process is not a program per se, but rather an artefact that performs some desired function. His reason for advocating this approach is to enable learners to recognize what programming has in common with other problem-solving tasks in order to facilitate transfer.

Another factor which may have led pupils to make this error is that they could have been applying two other, well-rehearsed schemata (from topics 3, 4, 6 and 7) i.e. *to output text, you use the PRINT command and put the text inside quotes,* and also, *to alter the output, you edit the program.* They needed to be reminded that these particular schemata don't work when the nature of the output should be determined in part by the input values, without the programmer having to alter the program, and for this reason the chosen intervention by the teacher was successful. It is in keeping with Polya's advice that the teacher's help should be unobtrusive.

There may have been less likelihood of this error occurring if, in task 6, the next two lines of the coding for 'output_poster' had been given, i.e.

```
2030 PRINT
2040 PRINT type_of_event$
```

However, it would not have been so easy to assess pupils' understanding of how variables work in the program, and for pupils to assess their own understanding.

Fragile knowledge in various forms (missing, inert, misplaced) exacerbated by strategic shortfall seems to provide a good explanation for this type of error having occurred (Perkins and Martin, 1986).

5.3.5 *The 'address cards' problem*

This is the first of two problems in topic 10. Topic 10 begins by reminding pupils of the design for 'posters' from task 9 (the version in which the user is

---

23 In Kafai's study (1993) an authentic context was created by getting pupils to create games software for a younger class of pupils to use, and through which the younger pupils could learn about fractions. The project involved evaluation sessions where the younger pupils joined the older ones while they were working on their games, and they could walk around looking at, trying out and discussing the various games with their designers.

Problem Description

When moving house you can send cards like this to friends, family and so on.

```
**************************

Please note our new address

Mr and Mrs J Pollock
25 Main Street
NEWTOWN
Scotland
JW5 6PH

**************************
```

In Tasks 1 and 2 you will design and write a program to produce printed 'change of address' cards like this.

The user should be prompted to enter the number of cards that are needed, and the details which should appear on each card. Note that each line of the address should be entered separately.

Figure 14: Topic 10: the problem description for 'address cards' (unit 2 task sheets p.46).

prompted to enter the number of posters required), and by explaining more fully about the top-down method of designing the program.

The 'address cards' problem is then introduced (figure 14). In task 1 pupils must *tailor* (Soloway, 1986) the design for the 'posters' program in order to produce the design for 'address cards'. In task 2 there is guidance on selecting data to test the program.

5.3.6 Pupils' progress with the 'address cards' problem

Most pupils faired better with this task, and indeed three pupils (Claudia, Bernadette and Lorraine) produced completely error-free solutions. The pattern of errors that occurred for task 2 and how long each pupil spent on topic 10 are shown in table 9 (overleaf).

Chris was permitted by the class teacher to omit task 2 since she wished him to concentrate on the design aspect of the task (task 1). As before, some pupils did not document their solutions fully, such as by failing to retain printouts in their folders. Obviously this creates difficulties for the teacher when conducting on-going assessment of pupils' work[24], but it creates even bigger difficulties for pupils since it hampers their ability to re-use elements of previous solutions or to review their learning. When required to adapt earlier solutions pupils may have to reconstruct them, and if they don't do this accurately this creates further problems.

In task 1 most pupils were able to adapt the design for 'posters' correctly, although a few forgot to amend some of the details so that, for example, they were still referring to the type of event rather than the person's name. Only one pupil, Chris, appeared to have difficulty with designing the program, and his first attempt consisted of a long list of detailed steps, which he subsequently scored out before writing out a new version with the main steps listed first (the top-level design) followed by refinements for two of the steps. Designing programs is something that Chris wasn't very confident about, as he revealed in his unit 1 and unit 2 questionnaire responses, and I explored the reasons for this during interviews with him. His difficulties with top-down design are discussed again in chapters 6 and 8.

---

24 The class teacher insisted that it was each pupil's responsibility to retain any printouts in his or her folder, and it is only for the purposes of this analysis that I have obtained any missing printouts by accessing their user areas. This was not done for pupils during the course.

'address cards'	Written coding (Task 2) X = error					Printed listing (Task 2)	Time on topic (periods)
	E1	E2	E3	E4	E5		
Bryan	X			X	X	Fixed no. of repetitions. [1]	5
Stephen							
Claire	X	X				Correct	5
Scott	X			X		Fixed no. of repetitions.	7
Paula		X		X		Correct	4
Claudia						Correct [1]	4
Carol-Anne	Incomplete					Fixed no. of repetitions.	7
Dawn	Incomplete					Missing prompt.	6
Chris							2
Caroline			X	X		Correct	6
Anthony	X					Correct [1]	8
Michael	Incomplete					Correct	9
Kirsty				X		Correct	6
Bernadette						Correct	7
Kelly	Incomplete					Correct	7
Catherine						Fixed no. of repetitions.	8
Kathleen	X					Correct	7
David						Poor user interface. [1]	5
Gerard	Answer booklet missing					Fixed no. of repetitions.	9
Lorraine						Correct	3
Frequency of errors	**5**	**2**	**1**	**5**	**1**		

Key:	Written coding
E1	Missing $
E2	Missing END PROC
E3	'OUTPUT' instead of 'PRINT'
E4	Variable or procedure name mismatch
E5	Output of a constant instead of a variable
	Printed listing
1	Not in folder, retrieved from pupil's user area.

Table 9: Pupils' progress on the 'address cards' problem of Topic 10 (Unit 2 task sheets p.47).

```
]RUN
THIS PROGRAM PROMPTS FPR [sic] AND INPUTS AN ADDRESS CARD
PLEASE ENTER THE NUMBER OF CARDS
?3
PLEASE ENTER THE MESSAGE

PLEASE NOTE OUR NEW ADDRESS

PLEASE ENTER THE NAME
?JONES

PLEASE ENTER THE ADDRESS
?...
```

Figure 15: Topic 10, the start of David's input screen for task 2.

In the *written* version of the program for task 2, far fewer errors were made than for the written version of 'posters', even though pupils had to write out the entire program rather than just a part of it. This may be because some pupils were working from listings of their 'posters' program from task 9 which they had got to work successfully. One would also hope that most pupils had learned something from their mistakes in the previous topic. For example, of the eleven pupils who had used a numeric instead of a string variable in the 'posters' program (task 7), only three repeated the error, although two other pupils made this error for the first time. Also the problem of including the data that the user should input to the program as part of the program itself (e.g. PRINT "Mr and Mrs Smith") has not recurred, although Bryan has this time placed the variable names inside quotes (exactly as Scott had done in 'posters'; as they sit next to each other perhaps he was looking at Scott's 'posters' solution at the time!).

When one examines their printed listings, Bryan, Scott, Carol-Anne, Catherine and Gerard appear to have based their programs on the version of the 'posters' program that they first produced in topic 9, rather than on the later version, since step 3 of their top-level algorithm for 'address cards' (to prompt for, and input the number of cards) has not been implemented. It may be that they each anticipated that the manner in which the solution would unfold for 'address cards' would mirror that of 'posters', i.e. that the solution would be obtained in two stages, firstly with the number of repetitions being set as a constant value within the program, and then later with the user determining the number of repetitions. In each coursework assessment task (pupils have already tackled four coursework assessments by this stage) when pupils are evaluating their solutions at the end, they are asked to read the problem description again to ascertain that their solution does do everything that it should do. This does not appear to have happened here, since otherwise they could have spotted the error; instead the process of testing the program seems to have ended whenever they got the program to run without producing error messages. Dawn also has an error in her program which, had she read the problem description again, she should have been able to spot (she has at one stage omitted a prompt so that only a '?' appears on the screen).

Apart from these errors, and some minor discrepancies between the internal commentary and what the program actually does (e.g. the internal commentary may refer to there being three address cards produced by the program when in fact the number can vary), the only other program which failed to meet the specification was David's. It did run, but the instructions to the user and the screen prompts were confusing (figure 15 – note that any

```
  100 // Program to output prize draw tickets
  110 // [pupil inserts name here]
  120 // [pupil inserts date here]
  130 MODE :=3
  140 tell_user_about_program
  150 PRINT "Please enter the number of tickets to be printed"
  160 INPUT no_of_tickets
  170 input_details
  180 FOR counter:= 1 TO 3 DO      ---------- (3)
  190   output_ticket
  200   PRINT
  210 NEXT counter
  220 END
  230 //
  240 //
  250 //
  260 //
 1000 PROC tell_user_about_program
 1010   // tell the user what the program does
 1020   PRINT "This program prompts you to enter the number of tickets
you want and what each prize is."       ---------- (1)
 1030   PRINT
 1040   PRINT
 1050 END PROC tell_user_about_program
 1060 //
 1070 //
 1080 //
 1090 //
 2000 PROC input_details
 2010   // get the user to enter each prize
 2020   PRINT "What is the first prize"
 2030   INPUT prize1                          ---------- (2)
 2040   PRINT "What is the second prize"
 2050   INPUT prize2                          ---------- (2)
 2060 END PROC input_details
 2070 //
 2080 //
 2090 //
 2100 //
 3000 PROC output_details                    ---------- (4)
 3010   // output the numbered ticket
 3020   PRINT "*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*"
 3030   PRINT     "MIDTOWN YOUTH CLUB PRIZE DRAW"
 3040   PRINT
 3050   PRINT "1st prize....."; prize1       ---------- (5)
 3060   PRINT "2nd prize....."; prize2       ---------- (5)
 3070   PRINT
 3080   PRINT "PRIZE DRAW NUMBER ";no_of_tickets      ----- (6)
 3090   PRINT "*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*" (7)
 3100 END PROC output_details
```

Notes: (1) - word split on screen;
(2),(5) - wrong variable type;
(3) - should be: FOR counter:= 1 TO **no_of_tickets** DO
(4) - wrong procedure name (also at 'END PROC')
(6) - should be: PRINT "PRIZE DRAW NUMBER "; **counter**
(7) - too few stars on bottom border

Figure 16: Topic 10, the 'prize-draw' problem (notes for teachers on unit 2).

values which the user has to enter are preceded by a '?'). One gets the impression that David had not visualized clearly what should happen at the input and output stage of the program, in spite of the fact that his top-down design was correct.

5.3.7 The 'prize-draw' problem

This is the last problem in topic 10. There is a problem description which pupils are asked to read carefully, and they then have to load a 'buggy' program 'prize', obtain a printed listing of it, and attempt to debug it (an annotated printed listing of 'prize' is shown in figure 16 opposite; the errors are listed in the approximate order that they might be detected during testing at the computer). Once they have done this, and they have saved the corrected version of the program as 'prize1' and obtained two sample runs, they then have to compare the listings and identify all of the bugs in the original program, and explain how they discovered and corrected each one.

5.3.8 Pupils' progress with the 'prize-draw' problem

For the most part pupils succeeded in identifying and correcting the bugs in the program. The most revealing aspect of their solutions is the manner in which they have described how they were able to discover and correct each bug in the program. There were quite striking differences in the level of explanations offered by pupils and in their ability to describe accurately and fully the changes that they had made. Also within their descriptions and explanations some occasional misunderstandings are revealed, even for those pupils who succeeded in correcting the program fully. It is also clear that some pupils were much more strategic in their approach than others, and that quite a wide range of strategies has been employed, mostly with success. However the discussion begins with the least successful attempts.

5.3.8.1 Carol-Anne's, Bryan's and Scott's solutions

Carol-Anne, Bryan and Scott were seated beside each other in class to do any reading or written work, although they worked individually at computers. Theirs are the only solutions where the bug, identified as (3) in figure 16, has not been corrected, according to the printed listings that they have enclosed in their folders, so that the program always produces three printed tickets no matter which number the user requests. Along with their printed listings, Carol-Anne has enclosed only one sample run in which she has requested three tickets and the program has produced this number, and Bryan has also enclosed only one sample run in which he has requested two tickets and the

program has produced three. However Scott has two sample runs, in which he has requested three then two tickets and the program has produced the correct number in each case. Either the printed listing in his folder is not the most recent version of his program, or he has altered the constant value at line 180 from '3' to '2' between runs (as he had done in his previous solution for 'address cards').

Carol-Anne and Bryan have failed to correct the bug, identified as (6), which results in the same number appearing on each ticket. However this is quite a difficult aspect of the task because this particular technique, outputting the value of the loop counter within the loop, has not been taught yet (a number of pupils asked for help to number the tickets consecutively, and word about how to do this spread around the class).

Carol-Anne has introduced a logical error by placing the 'input_details' procedure within the loop, so that she has had to enter the information for the ticket three times when she ran the program (this shows up on her sample run).

Their descriptions of how they discovered and corrected each bug are as follows:

> ***Carol-Anne's description***
>
> *180 and 190 were in the wrong order so I swapped them around.*
>
> *190 I changed from output_ticket to output_details*
>
> *2030 and 2050 were not proper variable names I added an underline and a $ symbol to them both.*
>
> *3050 and 3060 ditto.*
>
> ***Bryan's description***
>
> *180 and 190 were in the wrong order, I swapped them around. 190, I changed from output_ticket to output_details. 180, I changed to '1 to ticket do' from '1 to 3 do'. 3080 ...; no-of-tickets to something about counter. I added string variables at lines 3050–3060.*
>
> ***Scott's description***
>
> *180 and 190 were in the wrong order. 190 I changed from output_ticket to output_details. 2030 and 2050 I added a $ variable. I also added a string variable to the end of lines 3050 and 3060.*

These descriptions mostly consist of a list of changes to the program. Carol-Anne hasn't explained why she thought lines 180 and 190 were in the wrong order, and she thinks, erroneously, that a 'proper' variable name has to include an underline symbol and a string symbol. Bryan and Scott have copied the first

part of Carol-Anne's description (neither of them had in fact swapped lines 180 and 190), and Bryan's description includes other changes which do not appear on his listing. Perhaps he has also failed to include the most recent version of the program in his folder or he has copied them from someone else? Even if they had both succeeded eventually in debugging 'prize', the manner in which they have documented the solution would make it extremely difficult for either of them to reconstruct it accurately if they ever needed to later.

None of their descriptions gives any indication of the strategies that they have used to test and debug the program. I am reminded of Nisbet's and Shucksmith's observation (1984) that, 'For most people – even some university students – the method of learning is 'no method'; you expose yourself to the possibility of learning – by reading or listening or watching or sometimes just attending classes – and you hope that learning will take place. Of course it does, but often very inefficiently.' (p.5)

5.3.8.2 David's, Anthony's, Dawn's, Bernadette's and Kathleen's solutions

David, Dawn and Bernadette succeeded in correcting all of the errors in 'prize'. Anthony's and Kathleen's programs are correct apart from one bug (they haven't corrected the formatting of the text where a word is split on the screen). Bernadette and Kathleen have marked each correction that they made onto the original listing.

Their descriptions suggest that they have relied mostly, perhaps entirely, on testing the program at the computer. David's and Anthony's descriptions contain some errors (David's reference to 'output_ticket' and 'output_details' as variable names, and Anthony's observation that 'when it asks for second prize it doesn't allow for anything to be typed in'). However on the whole they do succeed quite well in describing most of the errors that they found, e.g. 'The prize draw number is the same as the amount of tickets (it doesn't change with each ticket).' The girls' descriptions are more general and they focus on the processes that they carried out to test and debug the program, although Kathleen's description focuses on only one type of error.

> ***David's description*** *The counter was set to the wrong number. Output_ticket was not the same variable name as output_details. Input prize should have been a string variable. The borders were different lengths. The prize draw number did not change.*

> ***Anthony's description*** *The word prize is spaced between two lines, P on the end of the first line and rize on the start of the second. When it asks for first prize it allows a number to be typed in. When it asks for second prize it doesn't allow for anything to be typed in. The prize draw number is the same as the amount of tickets (it doesn't change with each ticket). Only prints 4 tickets, top border is too long. There is no dollar signs in the program.*

Dawn's and Bernadette's descriptions *When I ran the program there were a lot of errors so I changed all the errors I could find until I had an error-free program.*

Kathleen's description *I ran the program and if it came up bad value I escaped and it told me which line the program had gone up to then I knew that there was something wrong with that line I corrected it then ran it again and kept on doing this till the whole program ran completely.*

5.3.8.3 *Michael's, Claire's, Caroline's and Claudia's solutions*

Claire and Claudia corrected all of the bugs in the program. Caroline has almost got the program working, apart from the correct formatting of output to the screen, which she appears to have overlooked (errors (1) and (7)). Michael has not managed to number each ticket consecutively.

There is evidence, from their written descriptions, of a range of strategies being used successfully which don't just rely on testing at the computer. Both Michael and Claire have begun by checking through the printed listing of 'prize', and Caroline and Claudia have referred to their previous solutions.

Michael's description *I got a printed listing and looked down it seeing what was wrong and corrected it from there.*

Claire's description *I listed to check if I could find any straight away without having to run it. I then kept listing it to find out and I kept trying.*

Caroline's description *I referred to the design for the poster and compared it to the printed listing I obtained of 'prize'. I corrected the bugs added in new lines in places and crossed out errors in others. I think it is very helpful to have a detailed design to work from.*

Claudia's description *Once I ran the program I referred to the example on page 41 of the task sheets [part of the coding for the first version of 'posters'] and changed everything in order. I kept doing this until my program ran with no bugs at all.*

5.3.8.4 *Lorraine's, Catherine's, Kelly's and Kirsty's solutions*

Apart from Kirsty's failure to adjust the output to make the borders for the ticket of equal length, they have each succeeded in removing all of the hidden bugs in 'prize'. They have written their corrections onto the original listing; Lorraine's description below corresponds to the order in which the changes appear on the listing, and Kelly has numbered the changes on the listing to correspond to her list of points.

All of their descriptions detail the individual changes they have made to the program and, apart from Lorraine's, why they thought each was necessary. (Kirsty's reference to the program not accepting numbers does leave me a little concerned about her understanding at that time of the different variable types.) They have used the correct programming terms throughout, e.g. point (6) of

Kelly's description, apart from Kirsty's references to the 'counter command' and 'prize commands'. In addition to detailing the specific changes they made to correct the program, Lorraine and Catherine have also indicated their general strategy.

Lorraine's description

First I looked over to see what looked wrong then I ran the program to discover what bugs there were.

1. *the counter – I changed the number to no_of_tickets*
2. *I changed the procedure name.*
3. *I added in a line where the whole sentence didn't fit in.*
4. *I added $ signs and I changed a variable name from no_of_tickets to counter.*

Catherine's description

To find the bugs I ran the program to see what happened, then began to change some of the procedures and some bits in the main program so that eventually when I ran the program it worked. I also looked at a previous program to see some of the things in it to help me find the bugs.

1. *When I ran the program the top line was all together so I split it into two.*
2. *I put in what the program does.*
3. *I put in printer commands.*
4. *I changed output_ticket to output_details because the procedures didn't match.*
5. *I had to use a dollar sign at the end of prize1 in the input details procedure to make it a string variable.*
6. *I had to do the same thing with prize2*
7. *I had to change the length of the border because it was too big.*
8. *I had to add $ sign to prize1 and prize2 in the output to make them the same as the input.*
9. *I had to put counter in so that I would get a different number on each ticket.*

Kelly's description

1) *I ran it and the writing was all in one line so I split it into two.*
2) *There was no printer commands so I had to type them in.*
3) *When I wanted to print it printed 3 each time so I changed the line so that it printed as many copies as I wanted.*
4) *I had to change output_ticket to output_details because the procedures didn't match it.*
5) *I had to add a line which told the user what the program was about.*

6) *I had to add a dollar sign on to the end of prize1 in the input details procedure to make it a string variable.*

7) *I had to do the same to the end of prize2.*

8) *I had to change the length of the border on the printed output because it was longer than the bottom border.*

9) *I had to add a dollar sign on to the end of prize1 and prize2 in the output so that it would be the same as the input.*

10) *I had to add the word counter in the output so that I could get a different number on each ticket.*

Kirsty's description

For the first one I noticed there were no printer commands at the beginning and so I slotted in VDU 2 and VDU 3 commands.

When I was trying to get 2 runs printed out I got 3 so I knew I had to fix the counter command. I changed line 180 by putting in an extra command.

I changed line 190 after none of my procedures were working because of variables being wrong.

I cut the sentence telling us what the program does as it went over one line and cut a word in half.

I had to add dollar signs to make the prize commands string variables so I could change the prize. I noticed this after my program wouldn't accept numbers.

I added the counter command to line 3090 as the tickets were not numbering themselves and the counter command corrected this.

5.3.8.5 *Paula's and Chris's solutions*

I have left the discussion of Paula's and Chris's solutions to the end because their solutions each exhibit distinctive characteristics.

Paula has corrected each of the bugs, as well as making improvements to the internal commentary by ensuring that each main step of the program is explained within it (corresponding to what would be each step of the top-level design).

Paula's description *I looked through the listing to make sure that all the procedures had the same name – output_ticket change to output_details. Make sure that 'string' was at the end of anything the user inputs. Added two '-*-*' at end of ticket. Ran the program. On ticket change no_of_tickets to counter and changed FOR counter 1 to no_of_tickets DO instead of printing 3 all the time.*

Paula's description, although less detailed than some of the others, reveals that she has checked systematically for certain types of errors on the listing before running the program. In so doing, she has anticipated some of the advice that is contained in the summary sheets for topic 10, which is designed to guide

pupils' subsequent debugging attempts. This advice was not presented in advance of pupils tackling the 'prize-draw' problem because we did not wish that the solution should be obtained routinely by applying a given algorithm; rather the problem should require some effort and thinking. Paula's solution demonstrates her 'planful' approach to learning (see Nisbet and Shucksmith, 1986, p.30).

Chris has corrected all but one of the bugs in the program (he has not adjusted one of the borders to make it the same length as the other).

> ***Chris's description*** *Counter said 1 to 10 not 1 to no_of_tickets. I found it because it printed 10 not 3. 1020 is too long prize is split up. I found it when I ran it. Prize 1 and 2 were not string variables. I just remembered that when bad value came up. Line 3090 should have counter not no_of_tickets.*

His description is, like Paula's, fairly brief. Within it he makes reference to a particular symptom – getting a 'Bad value' error message when he runs the program – which he is able to associate with a particular type of bug. It is evident that he is beginning to build a knowledge of symptom-bug associations which will enable him to debug his programs efficiently. Kirsty, in her description, also demonstrates her ability to make ready associations between particular bugs and their symptoms, ('When I was trying to get 2 runs printed out I got 3 so I knew I had to fix the counter command.').

Linn and Clancy (1992) highlight knowledge of common bugs and their associated 'fingerprints' as an important aspect of program design (the 'fingerprint principle', see 2.7.4). This and other of their design principles – recycling, reflection and literacy – are exemplified in pupils' solutions to Topic 9 and 10 problems.

5.4 REVIEW AND DISCUSSION

This section reviews the findings on progress in relation to three aspects: pupils' emerging expertise as programmers; the nature of the difficulties that they experienced with programming; and variations between pupils in their performances.

5.4.1 Pupils' emerging expertise as programmers

Pupils' performances in topics 9 and 10 provide a good summation of their learning at – for all but one pupil – an interim stage of the programming course, encompassing most of the key aspects that are taught within the first two units.

All pupils have demonstrated the ability to apply a top-down approach to design, although Chris experienced some initial difficulties with his design for 'address cards'. However the elaborated version of the design for 'posters' (where the user requests the number of posters) may not have been understood by Bryan, Scott, Carol-Anne, Catherine and Gerard, according to their solutions to topic 10 problems to which the same design template should apply. This is signified by their failure to implement the step in the design referred to above in the 'address cards' program, and/or by not detecting the bug in the 'prize' program which fixes the number of tickets to three.

The distinction between a numeric and string variable was still eluding some pupils in topic 10, although most seem to have grasped the difference by the time they reached the end of the topic, according to their printed listings for 'prize1' and their descriptions of how they managed to spot each bug in the original program and correct it. All pupils seemed to grasp the idea of using a loop in the program to repeat a set of commands. The most common error, linked to the problem identified in the paragraph above, was a failure to use a variable rather than a constant value to determine the number of repetitions.

The presentation of a sequence of related problems, using techniques of scaffolding and fading, has therefore been a successful strategy to enable most pupils to get to grips with difficult concepts.

When one examines the printed listings, all of Stephen's, Claire's, Paula's, Claudia's, Kelly's and Lorraine's programs are correct, and, apart from one or two errors in the formatting of the output to the screen in 'prize1' (identified as (1) and (7) in figure 16), all of Chris's, Caroline's, Anthony's and Kathleen's programs are correct. Dawn and Bernadette have each omitted an input prompt in one of their programs, and Michael has failed to number the tickets consecutively in 'prize1' (identified as (6) in figure 16) but apart from this they have made no other errors. Kirsty's program for the second version of 'posters' and the design do not match up precisely, and she has not adjusted one of the borders for the prize-draw ticket, but apart from this she has made no other errors. All of these pupils' programs are written in a readable style; they are modular, include sufficient internal commentary, and use meaningful identifiers (i.e. variable and procedure names). Their finished work (written designs and finished programs) is therefore judged to be of a highly satisfactory standard. However there is a considerable variation among this group in their rates of progress.

The other six pupils are Bryan, Scott, Carol-Anne, Catherine, David and Gerard. Their programs run without generating error messages, and they are also written in a readable style, but there is a logical error in at least two of each of their programs (apart from David's). I have discussed the errors in Catherine's programs at length; most of her difficulties seem to have arisen from using a constant value in place of a variable, although she has corrected the errors of this type in 'prize1'. Bryan, Scott, Carol-Anne and Gerard made the same error as Catherine in the 'address cards' problem (by using a fixed number of repetitions) but only Gerard's printed listing shows that he has fixed this bug in 'prize1'. David seemed to struggle when it came to implementing his designs (if indeed he attempted to implement them at all) and although he generally got his programs to work, of a fashion, there is evidence of 'tinkering' where parts of the program that were correct to begin with have been altered, e.g. he has removed the VDU 2 command (to send output to the printer) in 'posters'. Carol-Anne also managed to introduce a further bug in one of her programs, 'prize1', by placing the 'input_details' procedure within the loop. This group of pupils demonstrated a less secure understanding than the others. However, bearing in mind Perkins and Martin's (1986) observations on the extraordinary demands that programming makes on learners, it would be unfair to judge their work as being of an unsatisfactory standard. For the most part it captures their active attempts to make sense of complex subject matter.

Overall, of the 95 program listings that were scrutinized (including the first, introductory program in topic 9 on inputting a name and outputting it a set number of times), 73 (77% approximately) were completely correct, i.e. they met all the requirements that were stated in the problem description, 11 contained one or more logical errors, and 8 had at least one deficiency (often quite minor) in the user interface. All the programs ran without generating any error messages (unless one enters a string where a numeric is expected in the program – better input prompts would overcome this), and they are all written in a readable style.

Fragile knowledge of various kinds is exhibited through the errors that pupils made, together with some strategic shortfall, however the degree of floundering that other researchers have noted amongst beginner programmers and the over-use of trial and error strategies (as a substitute for planning what to do in advance) has not been exhibited in their work. Pupils did plan their programs and wrote them out before going to computers, and they succeeded relatively well in doing this, and not all of their debugging efforts took place at the computer. Already pupils are adopting a planned and systematic approach

to problem solving, although they are not all equally careful about how they go about their work (this aspect is discussed below).

5.4.2 *The nature of the difficulties that pupils experienced with programming*

While most pupils have achieved successful outcomes with their written designs and finished programs, nearly everyone has encountered some difficulties en route to obtaining solutions, apart from with the first, introductory problem in topic 9. When we come to examine the other sources of evidence, it can be seen that most pupils viewed this positively as an opportunity to learn from mistakes, and some considered that they had learned important and fundamental lessons (Caroline's review of the 'posters' problem from which I quoted earlier provides an example). However not all pupils expressed positive views, and some who made the most errors complained, paradoxically, about experiencing too much repetition. Scott's and Bryan's strategy of copying from some of Carol-Anne's work suggests that they did not feel very confident about their understanding, and that they were, unfortunately, intent on taking short-cuts. This may be indicative of a surface approach to learning.

Clearly some pupils in the class were not painstaking enough when it came to testing programs, since they had overlooked certain of the requirements that their programs should meet such as the correct formatting of output to the screen[25], and/or they had overlooked certain task requirements such as the number of test runs, the type of test data to use, and the nature of the evidence to be retained[26]. For some pupils the process of testing seems to end whenever they get the program to run without generating any error messages. They are failing to anticipate what the output from the program should be as they test it. By performing an on-going, systematic check of the problem description and the task specifications against their solutions, pupils could remedy these types of errors (there are tick-boxes for the pupil to indicate that a printed listing has been attached and numerous other prompts in the answer booklet to assist pupils). Checking back would enable pupils to anticipate better what ought to

---

25 The two bugs in 'prize' that were overlooked most frequently were error (1) - word split on screen (7 pupils), and error (7) - too few stars on bottom border (8 pupils). (See figure 17)

26 Nine pupils failed to include one or more of their printed listings and printed output in their folders; the worst offenders were David, with three listings missing, and Bryan and Claudia with two. David's practice of saving his programs using filenames of his own choosing and unrelated to the context (such as 'ATLAST', 'DICEY', 'ICANTDOIT') could explain why his listings are missing, because they would certainly not be easy to retrieve.

happen when they run the program, so that they are more alert to any problems. It may be that the timescale over which pupils develop solutions (resulting from having less than one hour at a time to work on them) and the intensity of their concentration on detailed aspects (e.g. while debugging) combine to make it harder for them to keep the overall picture in mind. Losing track of overall goals while working on detailed procedures, and failing to execute detailed procedures accurately while attention is focused on overall goals, are recognized difficulties facing novices solving mathematical problems (Schoenfeld, 1985).

Three of the learning strategies which Nisbet and Shucksmith (1986) highlight would appear to be pertinent to this situation; *monitoring* (defined as a continuous attempt to match efforts, answers and discoveries to initial questions or purposes); *checking* (a preliminary assessment of performance and results) and *self-testing* (a final self-assessment both of the results and performance on the task). Does the teaching emphasize these aspects sufficiently, beyond any specific advice within the programming materials about how to test and evaluate the finished program? This question is difficult to judge at this stage without having examined all of the available sources of evidence, including questionnaire returns and interview transcripts, and the answer may turn out to be different for each pupil (i.e. there may need to be more emphasis on these aspects for some pupils than for others). Perhaps the failure of some pupils on occasions to monitor, check and self-test is an indication that they lacked interest in the work and did not care about accomplishing it well (again there may be evidence of this from questionnaire and interview responses), or it may be that they felt under pressure to accomplish the work quickly. For example, pupils could try to save time by not filling in the progress record (which is an aid to monitoring) at the end of each period; by not checking over a solution (although there may be some corrections to do at the end of the topic, it is the teacher who will have to spend the time checking the work); and by failing to reflect properly on occasions when asked to review learning (self-testing). A good illustration of this is Scott's written review of the debugging task, 'prize', from review task 1 (Chris's review of this task is similar):

> *In this task I had to correct the program. I liked this task because I liked correcting programs. In this task I learned how to debug a program.*

Perhaps some pupils are receiving (seemingly) conflicting messages about, on the one hand, taking time to do the work carefully and ensuring that they understand, and, on the other, maintaining good progress with the course. If the routines of monitoring, checking and self-testing are squeezed out because

the pupil feels that there isn't time for them, then what happens when he or she reaches a genuine impasse with a problem (which is all the more likely to occur in this situation)? Is it better to take time to puzzle over the problem carefully, or to seek out the answer quickly from someone else?

On a much more positive note, in the written responses to the 'prize-draw' problem there is clear evidence of strategic thinking: planning (Paula), checking (Paula, Michael, Claire and Lorraine), and use of a heuristic strategy – working forwards from a previous solution (Caroline, Claudia and Catherine). There is also evidence that some pupils (Paula, Chris and Kirsty) are building a useful *knowledge base* of symptom-bug associations that will make them more adept at debugging. Where pupils have provided very detailed descriptions of how they discovered and corrected each bug, they generally made *correct and appropriate use of programming terms,* e.g. 'I had to use a dollar sign at the end of prize1 in the input details procedure to make it a string variable' (Catherine's 5th point). Not only has correct vocabulary been used, but also an effort has been made by most pupils to communicate the solution clearly through numbering or listing points, annotating the original listing and amplification, e.g. 'The prize draw number is the same as the amount of tickets (it doesn't change with each ticket)...' (Anthony's description). Some responses signify that pupils understood aspects of the problem and its solution that they had failed to grasp previously when working on similar problems (Catherine's response is particularly reassuring; she and Kelly collaborated on this task and they accomplished it well together; perhaps this partnership played a pivotal role in advancing Catherine's understanding).

5.4.3 *Variations between pupils in their performances*

There are considerable variations between pupils in their performances in these two topics; in the outcomes (their written work and printouts), the time to complete the work, and the amount of assistance sought. Pupils who made the fastest progress did not always produce the best work. Although there was no monitoring of how often pupils sought assistance, there is some evidence from other sources (e.g. question 9 in the unit 1 questionnaire) of how much importance pupils attached to getting help. Particularly in the later stages of the programming course when most pupils were working on unit 3 or unit 4, the class teacher would anticipate the points where certain pupils might have difficulties and she would monitor their progress carefully at these stages. She would coach individuals or small groups as they worked on a problem, and she would explain concepts and techniques that had not been understood. This was possible because many pupils in the class had, by this stage of the

programming course, become reasonably proficient and self-reliant, and they had also learned to collaborate and to assist each other effectively. My presence during some lessons also released the class teacher's time to provide in-depth support to those pupils who most needed it. This degree of targeted, individual assistance could not have been provided so effectively with more traditional 'chalk and talk' teaching methods, where part of the lesson would have to be given over to whole class teaching.

Whether pupils considered that the tasks in topics 9 and 10 were well matched to their abilities is explored in the interviews that took place after unit 2, where I discussed with pupils whether the extent of fading within topics 9 and 10, i.e. the withdrawal of additional help in its various forms, such as partial solutions, explicit guidance, hints, and opportunities for further practice and feedback, was at an appropriate pace for them. The findings on this are reported in chapter 7.

5.5 CONCLUSIONS

The main conclusions arising from the analysis of findings from records of progress and the folders of work are indicated below.

1. It does not seem feasible to cover the programming syllabus to credit level in anything like the recommended time of forty hours, unless the teacher schedules programming lessons without heed to pupils' competence or understanding, and/or 'teaches to the test' to enable pupils to demonstrate a limited set of performances by rote.

2. The majority of pupils have demonstrated a good understanding of programming concepts and problem solving processes and a considerable degree of programming competence. Many pupils' knowledge of string variables was fragile to begin with, but by the end of topic 10 the concept seems to have been grasped. Similarly a new template, introduced at the end of topic 9, proved difficult for a minority of the class to understand and apply. Over three-quarters of finished programs from topics 9 and 10 were error-free, and all ran without generating error messages and were written in a modular, readable and user-friendly style. Programs were produced following a process of analysis of the problem description and formulation of a written top-down design. There were very few errors in the designs (there was quite strong scaffolding for this aspect of their performance). Programs were then written out before being entered and tested at the computer. Program errors frequently pointed to pupils' active engagement with the problems, such as Catherine's buggy 'posters'

program. Only in a few instances is there evidence of 'tinkering' with a program to get it to work (however the evidence in the folders of work would underestimate the extent of this since they include only the final versions of programs). On the other hand, there is clear evidence of strategic thinking from pupils' descriptions of how they debugged the 'prize' program. The number and type of errors reduced as pupils progressed through the problems in spite of a reduction in the extent of cueing and hints, which indicates the success of the teaching strategy. From this I conclude that the 'average' student in the typically resourced classroom can learn to program well, given appropriate teaching. There is, however, one very important caveat. Individual pupils must be given whatever time they need to develop understanding and competence, even if this means that pupils do not all reach the same end point at the same time.

3. The majority of pupils seem to have managed their learning well, maintaining an adequate record of progress and an ordered and complete record of finished work. They applied a range of skills and strategies, including assessing task requirements, planning, designing, revisiting earlier solutions, anticipating the responses of the program, performing on-going monitoring, checking, and revising, and reviewing solutions at the end.

Some findings from this chapter are more tentative and raise further questions. There are many aspects of pupils' learning that have still to unfold.

4. How does one explain the differences in progress rates? Clearly a combination of factors interacted to produce this outcome, since no one factor on its own could account for the extent of the differences. Some possible factors emerged in the data – absences combined with incomplete records of progress, losing work (whether printouts, files or written work) and spending a long time on debugging. This last factor is probably very significant. In many non-computing environments errors may go undetected by the pupil and further progress is not automatically halted (although it may be hampered). In a programming environment errors generated by the computer must be dealt with before any further progress is possible.

5. Was the failure of certain pupils on occasions to monitor, check and self-test a consequence of their lack of effort or interest in the work, rushing to keep up with others in the class, or poorly developed metacognitive

skills? It is unclear, at this stage of reporting on the findings, whether there is enough emphasis on learning strategies beyond specific advice about program design techniques and how to test and evaluate a finished program. Perhaps the teaching on learning strategies is too narrowly focused on programming?

6 CASE STUDY FINDINGS II: ATTAINMENT

6.1 CHAPTER OVERVIEW

This chapter continues to report on the case study findings, but focusing on the evidence on pupils' programming attainments. The evidence is drawn from two sources, the class assessment record indicating the programming coursework grades awarded by the class teacher, and the assessment portfolios. The evidence from the assessment portfolios is used selectively, since it contains for each pupil several items of programming coursework accumulated over the course (for those pupils who were furthest ahead, it could contain up to ten items). The contents of the assessment portfolio do not differ in character from the contents of the folder of work, except that pupils worked unaided on assessments.

The evidence from the class assessment record is examined first to provide an overview of the results (section 6.2). Then selected items of coursework are examined to provide more detailed information on the performance of individual pupils and the class as a whole. There is an explanation of how the grades were derived from the evidence in the portfolios (section 6.3). The evidence is then reviewed and discussed in relation to two aspects: the efficacy of the assessment portfolio as an instrument for gathering evidence on summative attainment; and the quality of pupil learning, taking account of the earlier findings on progress (section 6.4). Finally the conclusions from this chapter are presented (section 6.5).

6.2 EVIDENCE ON ATTAINMENT (FROM RECORDS)

The class assessment record indicates that the following programming coursework grades were awarded by the class teacher.

Grade 1: Lorraine, Kirsty, Caroline, Chris, Paula

Grade 2: David, Kathleen, Catherine, Kelly, Claudia

Grade 3: Bernadette, Carol-Anne, Claire

Grade 4: Gerard, Michael, Anthony, Dawn, Scott, Stephen, Bryan

The better performance of the girls is the most striking feature of the results. Eight of the girls received credit grades in comparison to only two of the boys. The corresponding descriptors at each level are as follows:

> Credit Level (grades 1, 2)
> The candidate has demonstrated the ability in practical situations, given outline instructions, to analyse complex problems and design solutions, and to use computers to implement and evaluate practical solutions.
>
> General Level (grades 3, 4)
> The candidate has demonstrated the ability in practical situations, given detailed instructions, to analyse problems and design solutions, and to use computers to implement and evaluate practical solutions.
>
> (SEB, 1991, p.35)

How does one judge a problem's complexity to ensure that assessments can be matched accurately to levels? The extended grade related criteria (SEB, 1991, para 7 10, p.37–8) suggest that the following characteristics of complex problems can be identified (note that these are my inferences drawn from statements in the document):

(1) The problem is likely to require that pupils make inferences from the information available in the problem description;

(2) The problem is amenable to solution using a variety of methods, which should be weighed up;

(3) It should be necessary to seek out other information from a number of sources to implement the solution;

(4) It should be necessary to decide on procedures to be followed when using hardware and software;

(5) It should be necessary, when evaluating the solution, to judge contrasting views and formulate a balanced viewpoint;

(6) It should be possible to suggest improvements to the solution by applying the final outcome to wider contexts or by suggesting, with hindsight, an alternative strategy to achieve the final outcome.

These characteristics bear a striking resemblance to Resnick's (1987) working definition of higher order thinking.

6.3 EVIDENCE FROM THE ASSESSMENT PORTFOLIOS

6.3.1 Method of analysis

For each pupil, there are two items of evidence which were selected from the assessment portfolio as the best indicators of the pupil's normal level of achievement, in accordance with the SEB assessment requirements (in the

Topic Task(s) Level[1]	T7 4 F	T8 11 G	T11 8 G	T13 8–12 G	T14[2] 12–13 C	T15[2] 2–5 C	T16 8 C	T17[2] All C
Bryan			x	x				
Stephen	x	x						
Claire		x	[3]	x				
Scott			x	x				
Paula							x	x
Claudia				x	x			
Carol-Anne		x	[3]	x				
Dawn		x	x					
Chris							x	x
Caroline					x	x		
Anthony		x	x					
Michael	x	x	[3]					
Kirsty						x	x	
Bernadette			x	x				
Kelly				x	x			
Catherine				x	x			
Kathleen				x	x			
David				x	x			
Gerard		x	x					
Lorraine						x	x	
Frequency	2	7	6	10	6	3	4	2

Key
[1] LEVEL: F – FOUNDATION G – GENERAL C – CREDIT
[2] Evidence drawn from ordinary coursework.
[3] This assessment was by-passed.

Table 10: Evidence for each pupil which formed the basis for awarding the programming coursework grade.

programming project we took this to mean that the evidence would be drawn from among the pupil's most recent, successful work), and which therefore formed the basis for awarding the programming coursework grade. These items ranged from the topic 7 (foundation level) assessment to topic 17 (at credit level). It is very difficult to summarize across this evidence, for example in comparison to reporting the marks obtained by everyone in a common test, however the choice of assessment instrument does provide a very valid measure of pupils' achievements by demonstrating their ability to think in integrated ways about problems and solutions and to carry out solutions.

The two items of evidence that were selected for each pupil are shown opposite (table 10). The range of evidence at credit level reflects the extent to which pupils' progress had 'spread out' across topics 14 to 17 by the end of the course. Ideally, the evidence at credit level would have comprised the two coursework assessment tasks in unit 4 (within topics 16 and 18), however, as indicated earlier, this proved to be an impossible target. Instead 'ordinary' coursework was substituted where necessary if it was considered to be suitable in terms of the SEB requirements as outlined in their guidance document (SEB, 1993b), and also if there was evidence that it was substantially the pupil's own work. Some pupils were directed by the class teacher to by-pass the topic 11 assessment in order to accelerate their progress. Unfortunately for Michael, this was the wrong decision because he did not manage to reach the next general level assessment task. Had he done so it is possible that he could have got a better grade.

A selective and detailed examination of this evidence will enable assessments to be reached on pupils' understanding, competence, higher order thinking and metacognitive skills. The evidence will in addition provide a benchmark for comparison with other measures, e.g. pupils' self-assessments of their understanding and competence (obtained from questionnaire findings).

The analysis will be based entirely on finished work and not on any earlier attempts at solutions, and hence the hand-written version of the coding will be ignored. Since the coursework folder has already provided evidence of pupils' attainments in topics 9 and 10, I shall restrict the focus here to the assessment evidence from four of the later topics, specifically topics 11, 13 and 14 in unit 3, and topic 16 in unit 4. In so doing, I shall be able to examine one item of evidence for each of eighteen pupils in the class (if there are two items of evidence for a particular pupil I shall further restrict examination to the second item to avoid skewing). Hence the evidence to be examined in detail is as follows:

T11 (general level) assessment:	*Dawn, Anthony, Gerard*
T13 (general level) assessment:	*Bryan, Claire, Scott, Carol-Anne, Bernadette*
T14 (credit level) assessment:	*Claudia, Caroline, Kelly, Catherine, Kathleen, David*
T16 (credit level) assessment:	*Paula, Chris, Kirsty, Lorraine*

Each of the four assessments and the marking schemes which outline the assessment criteria for each item are included in appendix 2a. The assessment criteria relate to the problem solving schema which forms the basis for the assessment of the practical abilities element, i.e. to analysis and design, implementation, and evaluation.

In the following sub-sections, I shall present an overview of the analysis and design, implementation and evaluation skills to be assessed and highlight the progression in skills acquisition across the four items of evidence. Then the evidence will be examined for each pupil, beginning with the topic 11 assessment. Wherever possible I will discuss across a group of pupils' work to present the evidence in as succinct a manner as possible. Then the overall performance of the class will be discussed in relation to the items indicated in table 10.

6.3.2 *Assessing analysis and design skills*

The analysis phase precedes the design phase of the solution. The analysis (or problem definition) phase is concerned with understanding the problem in hand and working out what must be done. The design phase is concerned with how to do it (Dromey, 1982).

The problems that form the basis for assessment at general level are well defined and do not need much, if any, clarification. Hence the main requirement is for pupils to read the problem description carefully and to note accurately the requirements that must be met in the solution. As evidence that pupils have done this, they are asked to restate the problem in their own words. A further step is to analyse the problem to identify any inputs to the program and to describe the output it will produce (this can be set out on a screen design grid and/or described in words). Finally, pupils must produce a top-down design for the program.

Instructions

Use the log sheet to write down your answers. Remember to fill in the details at the top.

1. Describe the problem clearly and accurately in your own words.

2. Identify the inputs to the program. Use a Mode 7 grid to design the output screen format.

3. Write down the top level of the design, and then refine each of the steps as necessary.

   Use the design for the currency conversion problem to help you.

4. ...

Figure 17: Topic 11 assessment – the 'car rental' problem (unit 3 task sheets p.10–11).

Instructions

Either use the log sheet for your answers, or word process them. Remember to include the details at the top. Include all documentation.

1. Describe the problem clearly and accurately in your own words.

2. Identify any inputs to the program. Design the output screen formats (Mode 1).

3. Identify any of your library procedures that would be useful to solve this problem.

4. Find out about the EOD command, and how you can use it with REPEAT..UNTIL to enable the program to read and process the data until the end of the data is reached.

5. Design the top-level algorithm and refine any steps as necessary.

6. ...

Figure 18: Topic 16 assessment – the 'parents night' problem (unit 4 task sheets p.22–3).

Hence, in the topic 11 assessment (the 'car rental' problem) the first three steps of the solution process are as shown (figure 17). The problem description may go beyond a statement of user requirements to specify a language feature or technique that must be demonstrated in the solution (for example, the topic 16 assessment). This will have been done to ensure that pupils' performances in this aspect can be assessed, and it may also serve as a hint or clarification. When this is reiterated in the pupil's description of the problem, it should not be mis-read as an attempt by the pupil to 'gallop ahead' with the solution without first having taken time to understand the problem fully.

In the topic 16 assessment there are two further steps to be carried out as part of the analysis of the problem, firstly to identify elements of previous solutions which will be useful to solve the current problem, and secondly to analyse how a new technique can be incorporated into the solution (figure 18).

It could be argued that these two further steps amount to a consideration of resources for solving the problem rather than constituting part of the analysis, and that consideration of resources is an aspect of implementation. Accordingly, the order of the instructions to pupils should be altered so that the planning aspect of the task (step 5) precedes implementation (steps 3 and 4). However Soloway (1986) argues that one of the main stumbling blocks facing novices when they attempt to produce a top-down design is that they are asked to do this without recourse to any previously solved problems, and as a result they do a substantial amount of floundering and searching and tend to decompose a problem inappropriately. He states this more generally in terms of the learner's ability to transfer a solution strategy successfully:

> In teaching programming – and problem solving in general – a key objective is to develop useful methods of abstraction: If every problem a student must solve appears to be new and different, then there is little reuse of experience. A hallmark of expertise is the ability to view a current problem in terms of old problems, so that solution strategies can be transferred from the old situation to the current situation.
>
> (Soloway, 1986, p.852)

Linn and Clancy (1992) also stress the importance of building on experience and reusing ideas (enshrined as their 'recycling principle').

The very structured way in which pupils' design skills are extended within the course is designed to prevent them from floundering and from decomposing a problem in inappropriate ways. I shall use Soloway's (1986) language of plans (which are each associated with a particular programming goal) and his

Design for rainfall survey program

Top level

1 select mode 1

2 output the title, "Survey on rainfall – September"

3 set the total rainfall to 0

4 loop for each value of day from 1 to 30

5 read the amount

6 add the amount to the total to get the new value of total

7 end loop

8 output the total rainfall for the month

Figure 19: Topic 13 assessment – the starting point for subsequent modifications to the 'rainfall survey' program (unit 3 task sheets p.31).

analysis of the various methods of synthesizing plans (by tailoring, merging, nesting and abutment) to convey this.

In the topic 11 assessment, 'car rental', pupils must *tailor* an existing plan by working forwards from the design for the previous program. This is the method that pupils were directed to use previously in order to produce their top-down designs for the 'address cards' problem in topic 10.

In the topic 13 assessment there are five linked tasks, three of which require successive modifications to a program, the design for which is as shown in figure 19 opposite. This program computes the total rainfall for a month from values stored as data in the program, using an incremental total.

The first modification (in task 8) involves the *merging* of two plans – the 'total' plan with the 'tabular output' plan from topic 11 – so that the program also outputs a table, with appropriate headings, showing the amount of rainfall for each day. (This is the simplest way of arriving at a solution but there are of course alternatives, one of which is the *abutment* of the two plans – placing one plan end-on to the other – which would require two passes through the data, one to produce the table and the other to compute the total.) We do not draw pupils attention to the 'tabular output' plan from topic 11 – it is up to them to make the connection.

The second modification (in task 10) again involves the *merging* of two plans. The new plan, which is outlined to pupils within the task, involves the replacement of constant values in the program for the month and the number of days in the month, with variables whose values are read from data statements.

The third and last modification (in task 11) involves the *abutment* of plans, so that the program also calculates and outputs the average daily rainfall (there is no requirement to guard against division by zero).

In the 'guess the hidden number' problem (topic 14 assessment) an existing plan, associated with the goal of counting and outputting the occurrences of an event, must be *tailored*, and then a new plan (to validate user input – in this case the user presses the space-bar to see the next screen) must be *nested* within this to produce a solution.

In the 'parents' night' problem (topic 16 assessment) various existing plans (one of which is implemented as a procedure to centre text automatically on a particular row) must be nested within the design. Also pupils must *merge* the

new plan (to keep on reading and processing data until the end of the data is reached) within the solution.

A key design consideration is the *user-friendliness* of programs. Unit 1 focuses on careful layout of text and using colour. Unit 2 focuses on providing clear explanations (about what the program does and any output produced by the program) and instructions (about what to enter) on screen. Unit 3 focuses on producing tabular output, written user documentation to accompany the program, and validation of input. Unit 4 focuses on producing a title screen (giving details of author, version number etc.), producing a menu of choices that the user can select from, and using sound, graphics and special text effects.

Hence the criteria which are used to assess pupils' performances in 'analysis' and 'design' are as follows[27]:

Analysis: The pupil's written description of the problem should be in his or her own words, clearly stated, accurate and complete. All of the inputs to the program should be identified clearly. The output should be described clearly (in words, and/or by using a screen design grid). Elements of previous solutions that may be useful to solve the current problem are identified.

Design: The design for the program should: incorporate all of the required elements correctly (including user friendly features); be logically sequenced and modular; and be expressed clearly and refined in a top-down manner. Furthermore, the pupil should be able to discuss the problem and his or her chosen solution method.

Pupils will also be engaged in thinking analytically and in making design decisions at later stages of the solution process, e.g. when debugging ('Why am I getting this error?'), and when devising a test report ('Which test data should I select? What is the best way to present my test report?'). Linn and Clancy's (1992) choice of the term 'design skills' as an over-arching label to describe the application of general problem-solving skills to a programming context reinforces this point. However the over-arching use of the term 'design skills' is not adopted in this research because of the potential confusion that might

---

27 These are my interpretations of the extended grade related criteria (EGRC) for the practical abilities element in relation to 'Analysis of Problems and Design of Solutions' (SEB, 1991, para. 7 10 2, p.37).

arise with Standard Grade terminology, where 'design' is more narrowly associated with one phase of the problem solving process.

6.3.3 *Assessing implementation skills*

In Polya's (1948, p.12) terms, this is the stage where pupils carry out their plans accurately. He states:

> If the student has really conceived a plan, the teacher has now a relatively peaceful time. The main danger is that the student forgets his plan. This may easily happen if the student received his plan from outside, and accepted it on the authority of the teacher; but if he worked for it himself, even with some help, and conceived the final idea with satisfaction, he will not lose this idea easily. Yet the teacher must insist that the student should *check each step.*

In a programming context, this means that the pupil will be engaged initially in the process of translating his or her top-down design and any information contained on the screen design grids into a program, or, if the top-down design has been amended in some way, making the corresponding amendments to the program. We get pupils to write down the program (or the amendments to the program) first before going to the computer; in this way they do not have to deal simultaneously with encoding the design, and entering the program and correcting syntax errors. A key consideration is that the program should be easy to understand, modify and debug, and the use of procedures, meaningful identifiers (procedure and variable names) and appropriate internal commentary should help to bring this about. These are the features which most pupils would fail to include if they were to compose the program at the computer instead of writing it out beforehand (unless the program is very short and straightforward, or familiar already).

The presence of fragile knowledge (Perkins and Martin, 1986) will interfere with the process of translating the design into a program. Also an approach which is not systematic will hamper progress, e.g. the pupil may ignore the top-down design and attempt to construct the program line for line at the computer using trial and error. There may be a variety of reasons for choosing to use trial and error methods and these are examined later in the thesis. Even if a systematic approach is adopted, failure to monitor that the solution is on course as each step is put in place could result in errors and failure. Hence the first stage of implementation, to produce a written version of the program, is not without pitfalls.

The next stage, to enter and test the program, may also be fraught with difficulties, depending on the number of careless errors introduced when entering the program, and the number of errors already present in the written version of the program, which, as Perkins and Martin (1986) observe, can interact with each other to produce puzzling effects. It can be time-consuming as well to enter long programs, particularly when pupils do not have good keyboarding skills (this was a disliked aspect of programming for some pupils, according to their questionnaire responses).

It is very important for pupils to conceptualize the solution process as cyclical rather than linear. Any logical error in the program revealed by testing may have been introduced at a much earlier stage of the solution process, e.g. if the pupil's analysis of the problem was incorrect. It is then necessary for the pupil to retrace his or her steps and to reconstruct the solution.

Knowledge of symptom-bug associations and a methodical approach to debugging and testing are necessary to arrive at a correct solution efficiently; topic 10 is designed to reinforce these aspects. The outline instructions for each assessment task also encourage a methodical approach to testing.

In unit 4 topics, pupils must produce a test report as evidence of systematic testing. The test data will have been selected by the pupil to be representative of typical situations and extreme situations. For example, in relation to the topic 16 assessment, one of the library procedures which pupils will incorporate in the solution (to centre text on a given row) will have been tested with an input string of 40 characters (the maximum value), 0 characters (the minimum) and values in between.

The criteria against which pupils' performances in 'implementation' are assessed are as follows:

1. The finished program should correspond to the design; it should be easy to understand, modify and debug; it should be user friendly; and, most importantly, it should produce the correct output.

2. There is evidence of planning (e.g. to ensure that time at the computer is used productively), organization (e.g. by saving programs, retaining printouts and annotating them), monitoring of performance (e.g. to diagnose unproductive activities) and on-going testing. The pupil should be able to report on and discuss progress.

Topic 11 assessment

Does the program meet the specification?

Did you encounter any difficulties? If so, explain.

Can you suggest any improvements to the program?

Topic 13 assessment

(Task 10)

Contrast the different versions of the program (*survey5* and *survey6).* Which version is better, and why?

(Task 12)

Now get someone who is not in your computing class to use the program with your user documentation.

Describe any problems that he or she had.

Make any editing changes that are necessary to improve your user documentation.

Topic 14 assessment

(Task 13)

Why is the version which uses next_screen better?

Topic 14 and Topic 16 assessments

Does your program meet the specification?

Is it user friendly?

Is the program written in a style that makes it easy to adapt?

How could your solution be improved?

Figure 20: Prompts to assist pupils to evaluate their solutions.

6.3.4 *Assessing evaluation skills*

Polya (1948, p.14) highlights the importance of this phase of problem solving:

> Even fairly good students, when they have obtained the solution of the problem and written down neatly the argument, shut their books and look for something else. Doing so, they miss an important and instructive phase of the work. By looking back at the completed solution, by reconsidering and re-examining the result and the path that led to it, they could consolidate their knowledge and develop their ability to solve problems.

He suggests using the following prompts to assist students to 'look back' over their solutions: Can you check the result? Can you derive the result differently? Can you use the result, or the method, for some other problem? (ibid., p.14–5)

A wide range of similar prompts was used to aid the process of review (see figure 20). They were posed naturally. Since pupils tend to find evaluation difficult on their own, the prompts should also be posed verbally by the teacher, whenever possible, at the stage when the pupil has completed the testing process. A demonstration of the finished program to the teacher provides a natural opportunity for this to happen.

The assessment criteria which are applied to pupils' work under the heading of 'evaluation' are therefore as follows:

1. The pupil should be able to assess the adequacy of the solution by using test data and reporting on the results, and by comparing the outcome with the original brief.

2. The pupil should be able to suggest improvements to the solution or to the solution method or to the original brief.

3. The pupil should be able to consider ways of using the result, or the method, for some other problem.

The last criterion is very important to aid transfer. (Perkins and Salomon (1988) describe this as forward reaching, high road transfer.) There will be more likelihood that the result or method will come to mind when the pupil is searching later for ideas that he or she can re-use.

The review tasks and some questions posed during interviews also focused on aspects of evaluation, partly to compensate for the difficulty which arose in making time to discuss pupils' solutions with them individually.

6.3.5 *Evidence from the topic 11 assessment*

Anthony's, Gerard's and Dawn's topic 11 performances are discussed in turn.

6.3.5.1 *Topic 11 assessment – Anthony*

Anthony's description of the problem is in his own words and it mentions the inputs and the output produced by the program, although the format of the output (a table with two columns) is not clear from his description (he has also not included an output screen format).

> *I have to design a program in which the user has to type in a certain type of car and the price of its daily rental and it has to show the cost of renting the car from 1 to 14 days.*

His top-down design is correct, however he has not refined one of the steps from the top-level algorithm, which is the last step ('use a loop to output each row of the table'). It would have been appropriate to do this to keep the first level of refinements 'in parallel'.

His program has two minor errors – the second column of numbers is not centred beneath the heading and the numbers are not formatted to two decimal places[28] – however he claims that it does meet the specification. This, together with the fact that he did only one test run, indicates that his testing of the program was not very thorough. In his evaluation he has mentioned, in response to prompts, a difficulty encountered, and he has made a valid suggestion for improving the program.

> *Yes [the program does meet the specification]. I kept putting the symbols the wrong way around. It [the program] could give more details about the car (i.e. colour).*

6.3.5.2 *Topic 11 assessment – Gerard*

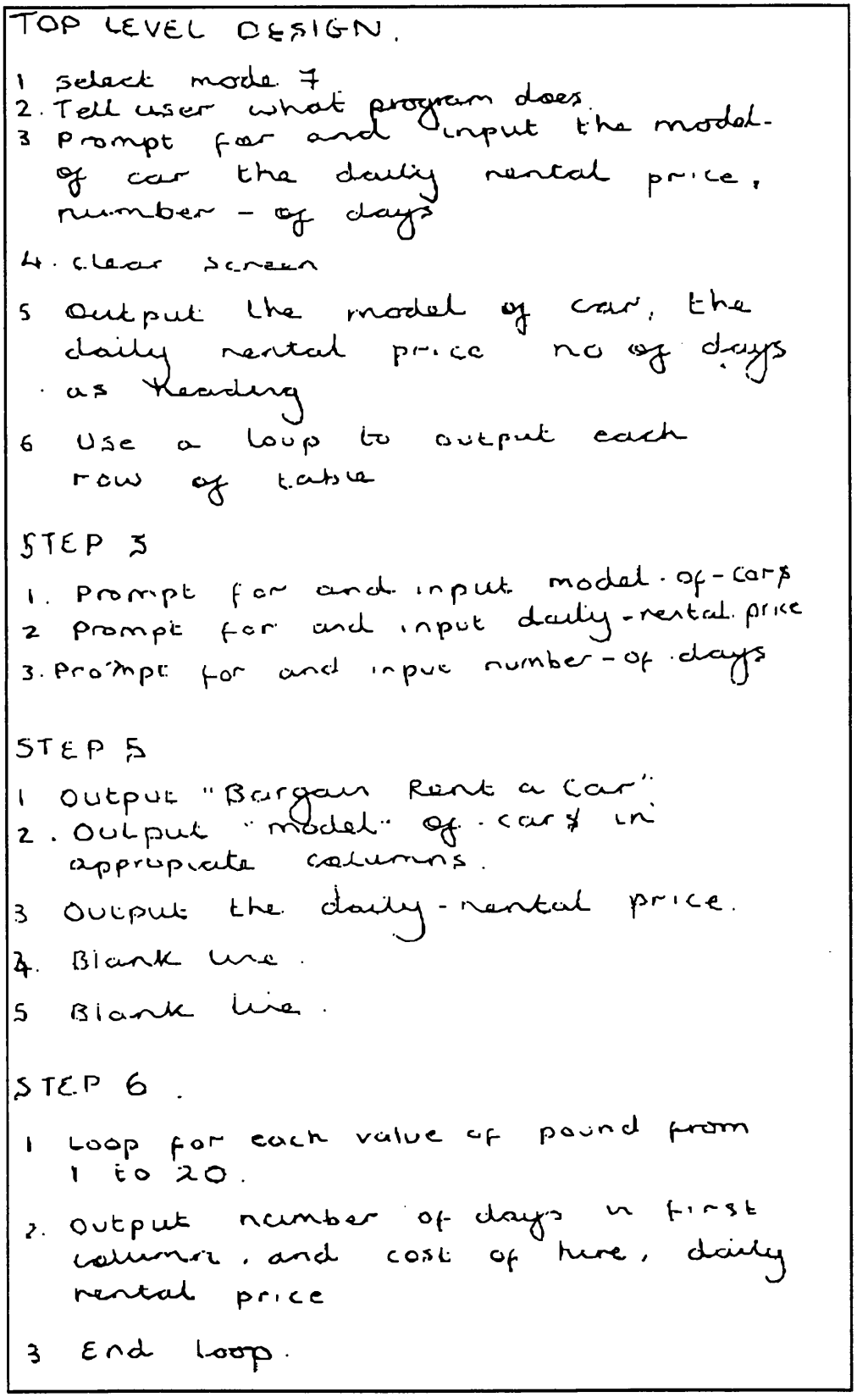
Gerard has also described the problem in his own words, however he has mentioned only one of the inputs (the model of car).

> *My problem is to design a program which gives you the price of a rental car for one to 14 days. I will have to design it so the user has to type in the type of car the output will look like a table.*

His top-down design is correct, however like Anthony he has not refined one of the steps from the top-level algorithm, which is the last step ('use a loop to output each row of the table').

---

28 The syntax to achieve this is very difficult – the numbers in the first column must be displayed as whole numbers and those in the second column as decimal numbers.



TOP LEVEL DESIGN.

1 Select mode 7
2. Tell user what program does.
3 Prompt for and input the model of car the daily rental price, number - of days
4. Clear screen
5 Output the model of car, the daily rental price no of days as heading
6 Use a loop to output each row of table

STEP 3

1. Prompt for and input model-of-cars
2 Prompt for and input daily-rental price
3. Prompt for and input number-of-days

STEP 5

1 Output "Bargain Rent a Car"
2. Output "model" of cars in appropriate columns.
3 Output the daily-rental price.
4. Blank line.
5 Blank line.

STEP 6.

1 Loop for each value of pound from 1 to 20.
2. Output number of days in first column, and cost of hire, daily rental price
3 End loop.

Figure 21: Topic 11 assessment – Dawn's top-level design.

His program has a minor error; he has failed to format the numbers in the second column to two decimal places. Also the comments in his program are not always accurate, which makes it confusing in places. For example,

```
2000 PROC input_values
2010     // THIS PROCEDURE PROMPTS THE USER TO ENTER THE
CAR MODEL
2020     PRINT "PLEASE ENTER THE RENTAL CHARGE"
...
```

This indicates that he has not examined closely or understood fully the specific functions performed by each part of his program.

He has failed to make any proper attempt to evaluate the solution, beyond testing it at the computer. All he has said is, 'Yes this program did meet the specification,' which in fact it did not entirely.

6.3.5.3 Topic 11 assessment – Dawn

Dawn's description of the problem sticks quite closely to the wording in the problem description, from which she has successfully extracted the main aspects. Her identification of the inputs and her output screen format are correct.

> *I will write a program which outputs a table on the screen showing the cost of renting a car between one and fourteen days.*
>
> *The inputs to the program are the model and the daily rental charge.*

Her top-down design (figure 21) contains a number of errors, some of which are misplaced elements from the previous solution. Although she has correctly identified that there are two inputs in her analysis of the problem, she has introduced a third input, 'number_of_days', in step 3 of her top-level design, and this is followed through into the refinement for that step (there were four inputs in the 'currency conversion' problem which preceded this one). As her first refinement of step 6 she has 'loop for each value of pound from 1 to 20' which is lifted directly from her previous solution. There are other errors, such as omitting to mention one of the headings in step 5, and a list of three outputs rather than two when she indicates which values will be output within the loop (the second refinement of step 6).

The evidence indicates that working forwards from a previous solution provides no guarantee of success for Dawn, although there are some aspects of her top-down design which are correct, such as the sequencing of steps. A careful analysis of the similarities and differences between the two problems is

a necessary precursor to producing a correctly tailored solution, and it is clear that her analysis has not been all that successful.

Her finished program is correct, however the internal commentary contains inaccuracies which make the program confusing to read (they are very similar to the example shown from Gerard's program). She did not 'cycle back' to revise the design once she had corrected the program, which resulted in some inconsistencies in her written documentation. Her evaluation includes an indication of a difficulty encountered and a suggestion for improving the program.

> *The program would look better if the headings were in colour. Yes the program does meet the specification. I had some problems trying to get the program to output the cost of hiring the cars to 2 dp.*

6.3.6 *Evidence from the topic 13 assessment*

Five pupils' work is examined here – Bryan's and Claire's separately, and Scott's, Carol-Anne's, and Bernadette's together.

6.3.6.1 Topic 13 assessment – Bryan

Because this is quite a lengthy assessment (extending over five tasks) there are more places to go wrong. Bryan's solution contains a number of what appear to be casual errors and omissions, however there are also parts of the assessment which he has done well.

The first modification to the solution (task 8) involves the merging of two plans – the 'total' plan with the 'tabular output' plan, so that the program also outputs a table with appropriate headings, showing the amount of rainfall for each day. Bryan has worked out the positions for the headings and for each column of figures on the design grid, and he has successfully merged the two plans in his design for the program, however in the program he has failed to output any headings for the table, and he has also omitted the details about the program at lines 100–120 (perhaps this is because he loaded this part of the program in an earlier task, and he has not checked it).

The next modification to the solution (task 9) is to edit the program and the data to output rainfall statistics for a different month (October rather than September). Bryan's program and sample run show only 30 days in the month (although the task outline includes a sample output screen showing 31 days).

The next modification (task 10) involves the merging of two plans by replacing two constant values (for the month and the number of days in the month)

Task 11

2 **Design for rainfall survey program - vers 5**

Top level

1 select mode 1
2. read month, days in month.
3 output the title.
4 set the total rainfall to 0.
5 output headings in columns.
6 loop for each value of day from 1to month do.
7 read the amount.
8 output days in month $ amount.
9 add the amount to the total to get the new value for total.
10 divide the total by the days in month to find the average.
11 end loop.
12 output the total rainfall for the month.
13. output the average rainfall for the month.

Figure 22: Topic 13 assessment – Bryan's top level design (task 11).

with variables whose values are read from data statements. Bryan has modified the design correctly, apart from omitting one change – his design still refers to the month of September. This time he has implemented all of the required changes to the program (but still no headings for the table). Since the task outline indicates that 'October, 31' should be the first values to be read from data statements, Bryan has added an extra value to the list of data items so that there are 31 rather than 30 rainfall statistics, however this did not alert him to the need to go back to correct his solution to task 9.

The final modification (task 11) involves the abutment of plans; the program must also compute and output the average monthly rainfall. First, a full and accurate description of the program is required. In his description he fails to mention the calculation and output of the total rainfall for the month, and there is no mention of inputs.

> *This program is able to output two headings eg. 'Survey on Rainfall' and the month eg. 'November' and a table with the amount of rainfall for each day for a month and then it outputs the average rainfall for the month.*

The top level of the design (figure 22) is almost correct apart from two places where he confuses the variable names (at steps 6 and 8). He computes the average within the loop, which is inefficient, however the efficiency of algorithms has not yet featured in the teaching. The program is correct (apart from the missing headings).

Lastly, in task 12, he produces a draft version of the user documentation (figure 23 overleaf), and a final version which is identical to the draft version. It is interesting to note that the two omissions in his earlier written description – mention of the total and of inputs – are included in the user documentation. Also he has included helpful 'internal commentary' (the explanations of the commands that the user has to enter). He has not discussed in writing how the user got on with testing the program and using his documentation. One could predict that the user would encounter some difficulties (for example, with how to enter the few simple details that he mentions under 'BRIEF DESCRIPTION'). He had, in fact, asked a friend to test his program and user documentation, but his friend was not interested in it and so Bryan did not get any useful feedback.[29] He reckoned that some of the later programs would have got a better reaction, like the quiz or guess the number game.

---

29 Interview 16, Scott and Bryan, 20/1/95, Transcripts p.198.

```
USER  DOCUMENTATION
===================

USING  THE  PROGRAM:  DESCRIPTION
-----------------------------
1. "Load survey8" press <RETURN>
2. Type "*PRINTER DES"
3. Type "VDU 2" press <RETURN>      (COMMANDS 3,4+5 WILL GIVE
4. Type "LIST" press <RETURN>       YOU A PRINTED LISTING)
5. Type "VDU 3" press <RETURN>
6. Type "VDU 2" press <RETURN>      (COMMANDS 6,7+8 WILL GIVE
7. Type "RUN" press <RETURN>        YOU A SAMPLE RUN)
8. Type "VDU 3" press <RETURN>

BRIEF  DESCRIPTION
------------------
This program calculates the total rainfall for the month entered and
it also calculates the average daily rainfall. All the user has to
do is enter a few simple details (rainfall each day in inches and
the amount of days in the month).

HARDWARE
--------
MONITOR
BBC MASTER
[SCHOOL] NETWORK

SOFTWARE
--------
COMAL
```

Figure 23: Topic 13 assessment – Bryan's user documentation.

6.3.6.2 Topic 13 assessment – Carol-Anne

Carol-Anne's greatest difficulty is with producing correct amendments to the design. She has labelled her printed listings and sample runs incorrectly (everything is out by one), and as a consequence her task 9 listing and sample run are missing. Her programs for tasks 8 and 11 contain some errors. She has put considerable effort into producing her user documentation.

In task 8, she has worked out the positions for the headings and for each column of figures on the design grid, and she has successfully merged the two plans in her design for the program. However her program contains three errors; the output of the headings for the table has been implemented within the 'output_title' procedure, which is not as she has indicated in her design; she has omitted the details about the program at lines 100–120; and she has forced a carriage return after the output of each figure in the first column of the table because she has used two separate print statements, 'PRINT TAB(8); day' and 'PRINT TAB(20); amount', to implement step 9 of her top level design, 'output day in 1st column and rainfall in 2nd.'[30]

The next problem is with her amendments to the design in task 10. None of the required amendments (to indicate that the month and the number of days in the month should be read into appropriate variables, and that the constant values 'September' and '30' should be replaced) have been made. Fortunately this does not seem to have hampered her ability to make the correct changes to the program, and she has fixed the error which resulted in the extra carriage return.

In task 11, she has described how she will amend her solution so that the program also computes and outputs the average monthly rainfall, but she hasn't actually answered the question, which is to give a full and accurate description of the program.

> *I will input a line to calculate the average daily rainfall for the month of November. There must be a calculation in this line. I will need to enter a line for the output of the average too.*

Again there are problems with her design for the program – the sequencing is incorrect, the control structure isn't indicated clearly (there is no 'begin loop'), and there are other omissions (for example, only one input is mentioned). In

---

30 The correct way to do it is to use one PRINT statement with a list of outputs, as follows:- 'PRINT TAB(8); day; TAB(20); amount', or to place a semi-colon after 'day' at the end of the first print statement. The semi-colon keeps the cursor on the same line.

```
TO LOAD THE PROGRAM
-------------------

1  PRESS THE <SHIFT> KEY AND THE <BREAK> KEY AT THE SAME TIME,
   LETTING THE <BREAK> KEY GO FIRST.
2  A BLUE MENU SCREEN WILL THEN APPEAR ON THE SCREEN, SELECT B TO LOG
   ON TO THE USER AREA.
3  ANOTHER BLUE SCREEN WILL THEN APPEAR, YOU HAVE TO ENTER THE USER
   NUMBER AND THE PASSWORD. YOU WILL HAVE TO PRESS <RETURN> BETWEEN
   EACH.
4  ...

TO RUN THIS PROGRAM
-------------------

AT THIS STAGE YOU SHOULD HAVE THE PROGRAM LISTING ON THE SCREEN. NOW
YOU WANT TO RUN THE PROGRAM.

1  KEY IN RUN AND PRESS <RETURN>.
2  ...
```

Figure 24: Topic 13 assessment – an extract from Carol-Anne's user documentation.

the program, she calculates the average by dividing the total by 30; apart from this there are no other errors.

Carol-Anne has tried out her program and user documentation with a user, and she has made a change to the draft version to produce the final version. An extract from her instructions to load and run the program is shown in figure 24 opposite. She is clearly able to construct a correctly sequenced, modular and readable algorithm to enable a user to carry out processes at the computer, so why has she such difficulty with constructing an algorithm to enable a computer to carry out a set of processes? Her ability to describe actual processes accurately and in detail has not transferred to designing programs. Perhaps her difficulty lies with visualisation; she needs a concrete model of the dynamic execution of a program to work with, so that she can build a robust understanding of how the program (and hence the algorithm) works.

DiSessa and Abelson (1986) describe a prototype programming environment, 'Boxer', which has been designed to make it easy for the user to implement simple ideas in highly visual ways. It is much more understandable than most current programming languages and has a 'familiar' feel. One of the features of Boxer is that you can change the value of a variable simply by altering the contents of a variable box that you have created on the screen, and if a program modifies the value of a variable, the contents of the box will be automatically updated on the screen. Also the hierarchical structure of a program can be represented by boxes within boxes. DiSessa and Abelson have therefore rejected some of the traditional requirements for programming languages, such as formal simplicity ('A better criterion is accessibility to a seven-year-old child'), efficiency ('...there are so many other important attributes of a language for controlling an interactive medium that efficiency must take a back seat.') and verifiability ('... rigor is not a primary requirement of an expressive medium.') (p.860). Other researchers (e.g. Thomas and Upah, 1996) have created computer simulations which allow students to manipulate simple algorithms and to vary inputs, so that they can observe the effects on any variables and on the outputs. Such programming environments may have assisted Carol-Anne (and other pupils) to develop a more intuitive understanding of programming.

6.3.6.3 Topic 13 assessment – Scott, Claire and Bernadette

Scott, Claire and Bernadette have made fewer errors than Bryan and Carol-Anne. In task 8, Scott hasn't actually used his layout from the design grid (the TAB columns are different in the program). They have all failed to include the

details (i.e. their name, what the program is about, and the date) at the start of the program, although Claire fixed this in her later versions of the program.

In task 9, both Scott's and Bernadette's sample runs show a table with only 30 rows, and only Claire has avoided making this error. The others may have failed to define the problem carefully before embarking on the solution, and as a consequence they have only taken on board one consideration, the most obvious one, which is that the month should change from 'September' to 'October' in the title. Providing a hint would make the task too simple. It might have been helpful to have presented earlier a different problem, in which pupils are led into a discussion of a range of considerations prior to design and implementation to heighten their awareness of the need to consider the problem from all angles before proceeding to design considerations.

In task 10, their amendments to the design are correct, apart from failing to indicate clearly that the title can mention any month. Bernadette has attached the wrong sample run for one of the programs, and Claire has accidentally overwritten one of the programs.

In task 11, Scott's description of the program is imprecise, however at least he has mentioned both inputs and outputs.

> *This program is able to output a table showing the amount of rainfall in a month. It is able to output this by reading data statements already entered by the programmer. It already outputs the average rainfall per month.*

Bernadette describes the output more clearly but fails to mention any inputs.

> *This program will output a heading with the month that is being used, output the headings Day & Rainfall for the table, display the table then the monthly rainfall in mm. then it will output the average daily rainfall.*

Claire gives only a partial description focusing on the final amendment to the program.

> *Output the total amount of rainfall for the month and the average daily amount by dividing the total amount by the number of days.*

Scott's top level design bears more than a passing resemblance to Bryan's; there is the same confusion of variable names and the average is computed inside the loop. It looks as though they have discussed the steps but then written them down in their own words. Claire's has three errors; there is a missing variable in step 2, ('2. read months'), the title still refers to a particular month ('3. output the title, 'Survey on rainfall – September'), and the last two steps are reversed (the average is output before the total). Bernadette's top level design is correct. There are no errors in any of their final programs.

They have each described the outcome of testing the program and user documentation on a real user. This is Scott's description.

> *The only problem he had was when amending the month he didn't know how to go to line 215 but he eventually found it by using the cursor. I should have added "by using the cursors" in the instructions.*

Unfortunately, he didn't add this instruction, so that his draft and final versions are identical, apart from the removal of the word 'draft'. Claire indicated that her user had problems, 'changing the months', however she also has not made any changes between the draft and final version. Bernadette reports a positive outcome:

> *The instructions seemed to be very clear as my friend had no difficulties at all.*

Their user documentation is well written and clearly set out. As with Bryan's and Carol-Anne's documentation, they have endeavoured to explain each step, rather than just give a list of instructions.

6.3.7 *Evidence from the topic 14 assessment*

The following pupils' work is examine – Claudia's, Caroline's, Kelly's, Catherine's, Kathleen's and David's. I shall look across the evidence, beginning with a discussion of pupils' work in task 12 and then moving on to discuss their work in task 13 and their evaluations.

The first part of task 12 – designing the screen layouts – posed no difficulties for pupils. However no-one produced an entirely error-free top-down design (the best solutions were Caroline's and Catherine's with only one error).

- Caroline and Kathleen omitted the step to generate a number at random;
- David's design did not indicate how the number of guesses would be computed, and Kathleen set the counter to zero within the loop so that each time the loop was obeyed it would be reset to zero;
- David and Claudia did not indicate clearly which steps had to be repeated (the exit condition is implied), Catherine failed to specify the exit condition ('14. Until'), and Kelly's design made no reference to repetition at all ('11. If number = hidden number tell user to go to screen 3.');
- There are some occasional lapses in aspects relating to the user interface (e.g. Claudia's design makes no mention of clearing the screen after the user has guessed correctly).

There are also a few lapses into coding in Claudia's, Kelly's and Catherine's designs (e.g. Claudia has, '6. Hidden number = RND(1,100)').

The main respect in which this problem differs from the other problems in topic 14 is that the program is interactive, whereas all of the other programs relied for input on values which were randomly generated within the program. Hence it is not closely analogous to any of the previous problems in topic 14, although it does relate to the same goal (to count and output the occurrences of an event), and there is a requirement to generate a number at random. It may be for this reason that most pupils found the task of designing the program to be quite challenging; it required more than the tailoring of an existing algorithm.

Fortunately most of the errors listed in their designs are not present in their finished programs. Only David's program contains a logical error, which is unrelated to the two errors in his design. He has within his main program a redundant section of code (from lines 223–60):

```
215 IF guess > hidden_number THEN
216   PRINT TAB( 9, 36); "YOUR GUESS WAS TOO HIGH"
218 END IF
220 IF guess < hidden_number THEN
221   PRINT TAB( 9, 36); "YOUR GUESS WAS TOO LOW"
223   IF guess > hidden_number THEN
224     ...

260   END IF
265 END IF
```

Since lines 221–260 will only be obeyed if the condition at line 220 is true, i.e. the value of guess is less than the hidden number, then the condition at line 223 will always evaluate to false. Clearly his solution has come about through a process of trial and error, but also enquiry – he has had to find out about the form of the IF statement which permits more than one statement to be obeyed if the condition is true. His solution is reminiscent of his earlier solution to the 'address cards' problem in topic 10 (see 5.3.6).

Any other deficiencies in their programs relate to the precise wording and layout of instructions on the screen (their screen design grids seem to have been forgotten about). Claudia's program is one of the worst offenders. Her screen 2 format shows the prompt to 'press the <space-bar>' at the bottom, and her screen 3 format shows the title for the program ('Guess the hidden number') centred and underlined at the top, and beneath this the words, 'You took __ guesses to guess correctly.' However this is what happens when you run her program.

```
...
Pick a number between 1 and 100
?52
Your guess was Correct
You took 5 guesses
```

Why did most pupils deviate from their layout plans? Claudia's finished program could hardly be described as an improvement on her original ideas, so it seems unlikely that any aesthetic considerations have led her to make changes to her original plans. It may be that pupils viewed how well they presented information on the screen as being less important than getting their games to work, and that the latter consideration took up most of their attention and energies since it was harder to think out.

In response to part (5), they all suggested systematic search strategies; Kelly, Catherine, Kathleen and David described a binary search (first guessing 50, then 25 or 75, depending on whether 50 was too high or too low, etc.), and Claudia and Caroline described going up or down in even steps of 10, then 5, then 2, thus narrowing the range.

Task 13 should have been straightforward since it required only accurate observation and an appreciation of the factors to consider when designing the user interface. When comparing the version of the program which uses 'next_screen' with the version which uses only the CLS command, Kathleen and Claudia were clear about the advantages.

> ***Kathleen*** *It [the version which uses CLS] jumps onto the next screen without giving enough time to read the first screen.*

> ***Claudia*** *[next_screen is better] because it gives the user a chance to read the instructions and understand the game. CLS clears the screen just too fast to be able to read.*

Only David seems not to have grasped the difference between the two effects; he gives as an advantage of the 'next_screen' procedure that, 'It instantly clears the inform_player screen,' (which indeed it does when the user presses the space-bar as directed, but so also does the CLS command).

Everyone, apart from Claudia, has produced good written evaluations. In the evaluations, they were asked to make reference to a range of programming principles (correctness, user-friendliness and ease of modification) as well as to suggest improvements to their solutions.

Kathleen's evaluation is satisfactory on all counts:

> *My program meets the specification. It does everything that the specification tells it to do [she provides two sample runs as evidence].*

It is user friendly. It gives instructions for everything that has to be done and tells the user when to enter a number and when to press the space bar.

It is relatively easy to adapt although one thing that makes it a little harder is that it is a long program with a lot of procedures.

It could be in colour. Or it could ask at the end of the program whether they want to play another game or not.

Caroline's evaluation falls down on only one criterion – correctness; her first suggestion to improve the solution was in fact a requirement that she hasn't met.

I think it does meet the specification [she provides one sample run as evidence].

I think my program is user friendly as it gives easy-to-follow instructions and is simple and clear.

The program uses procedures and is set out quite well. I think this makes it easier to understand and to adapt.

My solution could be improved by making the statement which tells the user how many guesses they took, appear on a new screen. I could also use coloured text to make it more attractive to the user.

The criterion which pupils seemed least sure of was the ease of adapting the solution ('Is the program written in a style that makes it easy to adapt?'). They are all in agreement that the program is easy to adapt (Kathleen qualifies this slightly), but Kelly, Claudia and Catherine don't explain why they think this is so. David suggests two possible adaptations, but he doesn't make any comment about the style in which the program is written:

The program would be simple to adapt to a heads or tails game or a guess the hidden number game from 1–200 by changing the random.

Caroline has mentioned its use of procedures and the layout of the program as features which make it easier to adapt. On the other hand, Kathleen has mentioned the length of the program and the number of procedures within it as factors which make it a little harder to adapt. Ideas surrounding readability were discussed during interviews, prompted by a number of pupils having indicated that they were uncertain about how to write readable programs (unit 2 questionnaire, question 6 part f). During interviews, we discussed what is meant by a readable program, which techniques can be used to make a program more readable, how it affects the ease with which you can adapt a program, and why it is important to be able to adapt a program easily. A number of pupils seemed to be confusing readability with user-friendliness. They thought that including comments in the program and using meaningful identifiers would make it more user friendly, perhaps because they envisaged that any users of

their programs would be doing the same sorts of things with the program that they were learning to do in the programming class. They didn't seem to realize that they had specialist expertise not possessed by 'ordinary' users.

Perhaps some of the factors that affect the readability of the program did not come to mind because pupils' programs are, generally, written in a style that makes them easy to understand, and therefore easy to modify. They have been taught to consider the layout of the program, to include comments and to use meaningful identifiers as a matter of course. Also some user friendly features of the program (such as screen prompts and on-screen explanations) do result in a program which is more easily understood, since these are incorporated into the program. Their programs are not, at this stage, very long (although Kathleen is beginning to think that they are!). Pupils therefore have no direct experience of trying to understand or modify a very lengthy and complex program which is written in an unreadable style, from which (painful) experience dawns the realization of how important it is to consider the style in which the program is written. Hence the discussion that took place during interviews may not have been as meaningful for pupils as I had thought, and the main points that emerged from it would not have received much subsequent reinforcement in practice. In the commercial world of software development, being able to produce easily modified programs is a major economic consideration, and the reliable performance of software through numerous upgrades depends upon it.

Claudia fails to provide any justification for her assessment of the user-friendliness of the program, stating simply, 'I feel that it is very user friendly.' She neither lacks the ability to provide good explanations nor the relevant understanding to make an appropriate response, as can be seen earlier in her response to task 13. There is clearly a need to frame the evaluation questions carefully to ensure that all pupils are aware of the need to provide proper justification for their views (a 'yes' or 'no' response won't suffice). The intended teaching input (when the pupil demonstrates his or her solution to the teacher and discusses it prior to producing a written evaluation) may be insufficient on its own to prompt full responses, and if the teacher's time is tied up it may not take place at all.

Kafai (1993) used focus sessions to initiate discussions around issues and ideas relevant to all of the pupils' programming work. The content and sequence of these sessions is not planned in advance but is arranged through discussion with pupils. This approach looks promising to reinforce and complement individual interactions, since it accords to pupils the responsibility to set the

agenda and to take their learning properly in hand, and it has more immediacy than the interview discussions which took place after the end of each unit where it was clear that some pupils had forgotten some aspects of their earlier work.

6.3.8 Evidence from the topic 16 assessment

The following pupils' work is examined here – Paula's, Chris's, Kirsty's and Lorraine's. Again I shall look across the evidence for this assessment.

In the topic 16 assessment (the 'parents' night' problem), there are four aspects to the analysis; firstly the pupil describes the problem in his or her own words; secondly the pupil identifies any inputs to the program and designs the output screen formats; thirdly the pupil identifies any library procedures that he or she has written which will be useful to solve the current problem; and lastly the pupil investigates how a new programming goal (to read and process until the end of the data is reached) can be realized in the solution.

The evidence from pupils' written descriptions and their screen formats indicates that they have understood the problem, although Kirsty's description and Lorraine's (which is not shown) stick rather closely to the original wording. All their descriptions are accurate, however the ordering of information could be improved in some cases. The inputs are not identified clearly; Paula supplies insufficient information (how many inputs, what type?), and Chris, Kirsty and Lorraine have not labelled the inputs separately.

> ***Paula's description*** *I want to design and write a program which could be used at a school parents night by parents. They should be able to find the teacher they want, the subject they teach and which room they will be in.*
>
> *The only inputs to the program will be the data statements.*
>
> ***Chris's description*** *Parents are getting lost at parents night. I have to design and write a program to direct the parents to the classes. The data will be name, subject and location. These must be read from data statements. There should be a title screen and a welcome screen. All text should be automatically centred. The parents will only have to press the space-bar.*
>
> ***Kirsty's description*** *I have to design and write a program which would help parents find teachers at a parents night. The data for each teacher has to give the name, subject and location. These should come from data statements when the program is run. I will be able to change the number of teachers without changing the program. A title screen at the start and a welcome screen will be needed and all the lines should be centred automatically. The user will have to press the space bar to get the page to change.*

The task of identifying library procedures that can be used in the program is straightforward; at this stage they have saved only four procedures to the library! All of their designs reflect accurately the information that they have

shown on their screen design grids. Any detailed information about wording or layout has been confined to refinements of the top-level algorithm, apart from Chris who has failed to create a new module to display the information on each teacher, so that his top-level algorithm is lengthy and contains much detailed information.

```
1   Change to mode 1
2   display a title screen
3   display an introduction
4   repeat
5     centre "[school name]" on the screen
6     centre "4th Year Parents' Night" on the screen
7     read the name, subject and class of teacher
8     ...
```

The details look out of place in comparison to steps 2 and 3, which he has refined separately (one of his library modules is 'title_screen', for which there is an existing written algorithm); he has failed to keep the levels of refinement 'in parallel'. This is precisely the same kind of difficulty that he experienced earlier when designing the 'address cards' program in topic 10.

In contrast to the solutions presented earlier to the topic 14 assessment, any design decisions have been followed through accurately into the finished program, apart from where Kirsty and Lorraine have nested 'next_screen' within 'teachers_page' (which is the module they have created to display the information on each teacher) in their designs, but not within the program. (It is clear that Kirsty and Lorraine have conferred on aspects of their solutions). All pupils provided appropriate evidence of testing in the form of a description of the test data sets (two were required) and sample runs.

Paula's written assessment of the correctness of her solution is better than the other pupils'. She has enumerated, without prompting, the features of her solution which demonstrate to her that she has met the task requirements.

> *My program does meet the specifications given. It gives the name, subject and location of each teacher. It reads from data statements which can be changed and the number of teachers can be varied. The text is centred and there is a title and welcome screen.*

One has the impression of a logical and rigorous mind at work; her earlier description of how she debugged 'prize' (in 5.3.8.5) reinforces this impression.

Rather disappointingly, when pupils came to assess the user-friendliness of the program, Chris seems to have confused readability with user-friendliness, and neither Kirsty nor Lorraine provide any justification for their assessments (of '...very user friendly').

Topic Task(s) Level[1]	T7 4 F	T8 11 G	T11 8 G	T13 8–12 G	T14 12–13 C	T15 2–5 C	T16 8 C	T17 All C	Prog. coursework grade[2]
Bryan			73%	72%					4
Stephen	93%	71%							4
Claire		93%		80%					3
Scott			80%	84%					4
Paula							96%	93%	1
Claudia				80%	71%				2
Carol-Anne		71%		76%					3
Dawn		64%	73%						4
Chris							80%	97%	1
Caroline					88%	74%			1
Anthony		64%	67%						4
Michael	86%	100%							4
Kirsty						95%	80%		1
Bernadette			87%	84%					3
Kelly				85%	76%				2
Catherine				85%	88%				2
Kathleen				76%	88%				2
David				70%	65%				2
Gerard		86%	67%						4
Lorraine						74%	76%		1
Average	90%	78%	75%	79%	79%	81%	83%	95%	

Notes
[1] Level: F – Foundation G – General C – Credit

[2] Grades:
1 / 2 – High/Satisfactory Overall Standard (Credit)
3 / 4 – High/Satisfactory Overall Standard (General)

Table 11: Pupils' percentage marks and grades for programming coursework.

Paula *My program is user friendly. All the user has to do is keep pressing the (space-bar).*

Chris *My program is user friendly as it contains comments for commands that are not too obvious.*

All pupils viewed their solutions as being easily adapted. Three of them focused on the ability to run the program with different data. Chris, like David in the topic 14 assessment, makes a suggestion for a possible adaptation to the program rather than commenting on the style in which the program is written.

Paula *The program can be adapted easily by changing only the data statements.*

Chris *The program could be adapted to help someone find another person in a large building...*

Kirsty *The style I used to write the program makes it easy to change for different data and to alter the procedures within the main program.*

Lorraine *The method [sic] in which I wrote the program makes the data easy to change.*

They have made appropriate suggestions to improve the solution.

Paula's suggestion *My solution could be improved if at the end of the data statements the program began again and if it was in colour.*

Chris's suggestion *My solution could be improved by telling the user where the room is i.e. room 21 is down the corridor.*

Kirsty's suggestion *I could improve my final solution by adding graphics or colour to make the screens more interesting.*

Lorraine's suggestion *...add colour and graphics.*

6.3.9 *Percentage marks and grades*

The percentage mark awarded to each pupil for each of the two assessment items, and his or her programming coursework grade are indicated opposite (table 11). All of the assessments were marked and graded twice by applying the standard marking schemes: firstly by the class teacher, to meet the formal assessment requirements for the course; and later (after she had submitted the coursework grades to the SEB) by me independently as part of the detailed scrutiny of the evidence for this research. For any aspects which are difficult or impossible to assess from the written evidence and printouts alone, such as discussions with the class teacher on progress with implementation, I relied on the class teacher's assessments. The marks shown in the table are the outcomes of the *second* marking process. Three examples of how I applied the marking

criteria to the pupils' work are provided in Appendix 2b. None of the marks that I awarded would have resulted in a different summative grade for any pupil, except for Scott, where the class teacher took account of the amount of help that he received from her when allocating his grade[31].

The marks ranged from 64% (Dawn's and Anthony's performance in the topic 8 assessment) to 100% (Michael's performance in the topic 8 assessment), with the average mark being 80.2% (it must, however, be borne in mind that this is an average of performance across a range of different tasks). The marks are distributed fairly evenly across the eight tasks, i.e. there is no clear trend towards better or poorer performance in the later assessments, except in the situation where only two pupils' work is being considered, in the topic 7 and topic 17 assessments. For the eighteen items of evidence which were examined in detail (in sub-sections 6.3.5–6.3.8), the average mark was 78.4%. Therefore the standard of performance in these items is fairly representative of the overall standard across all forty items.

Pupils' performances in 26 of the 40 assessment items (65%) merited the award of the upper grade at the relevant level (using a cut-off score of 75%) and the remaining performances merited the lower grade (using a cut-off score of 50%). The summative grade for programming coursework is mostly commensurate with the marks, however the class teacher had to decide on how to deal with 'boundary' cases, that is, pupils whose evidence comprised one foundation level item and one general level item, or one general level item and one credit level item, since there are no 'official' guidelines on dealing with this situation. She decided (i) to use the second item (their more recent work) to determine the level of award, and (ii) to restrict the availability of the upper grade to pupils who had completed *two* assessments at that level. Therefore, to get a general grade, pupils would need to have finished topic 8 (in unit 2) and have reached a satisfactory standard, and to get a credit grade, pupils would need to have finished unit 3 and have reached a satisfactory standard.

Any work which is the product of collaboration between pupils is not deemed to be acceptable as evidence for summative assessment purposes (SEB, 1991, para. 6 5 2, p.26). Pupils embarked on assessments whenever they were ready to, and at any given time some pupils were doing assessments while others were doing 'ordinary' work. There was no possibility of supervising them closely to ensure that they did not discuss the work (and they could do this

---

31 'When assigning the Coursework grades, due account should be taken of the extent and regularity of help given to the candidate.' (SEB, 1991, para. 6 6 3, p.28)

after class in any case). The class teacher therefore adopted a sensible approach by reminding pupils to do the assessments on their own, and if she was suspicious about copying she questioned pupils to assess their understanding of the work. Collaboration is an essential aspect of software development in real programming environments. It should be possible to design assessments to reveal the individual's contribution to the work.

6.4 REVIEW AND DISCUSSION

This section reviews the findings under the following headings: the efficacy of the assessment portfolio as a means of gathering evidence on summative attainment (6.4.1); the overall standard of performance in summative assessments (in 6.4.2); and factors affecting pupils' performances, and the importance of active learning (6.4.3).

6.4.1 The efficacy of the assessment portfolio as a means of gathering evidence on summative attainment

Although the assessment portfolio provides very good evidence on pupils' summative attainments, it may enhance pupils' learning to separate formative and summative assessment purposes by deriving the summative grades differently. There are three reasons for making this suggestion:

(1) The majority of items contained in the portfolio may not be taken into account when deriving the summative grade. This does not mean that pupils efforts have been wasted – all of the assessments resulted in formative feedback to the pupil – but it did limit opportunities for pupil collaboration.

(2) Although the assessment portfolios of pupils who completed unit 3 contained many items of evidence, it was still necessary to substitute ordinary coursework for one or both items to enable pupils to gain credit grades. Also the class teacher had to make a decision about how to deal with boundary cases where the first item was at one level and the second item at the level above.

(3) Opportunities for pupils to assess their own performance were restricted because the marking criteria cannot be revealed to pupils (SEB, 1993a).

Administering two summative assessment tasks at the end of the programming course would help overcome the restrictions (a choice of problems could be made available). The assessment portfolio could continue to be used for formative assessment. There would be less anxiety for pupils about

its contents, there would be no need to restrict pupils to working on their own, pupils could choose the items that they wanted to put in it, and pupils could assess their own performance or that of other pupils by formulating and/or applying a range of criteria to their work, under the teacher's guidance.

6.4.2 *The overall standard of pupils' performances in assessments*

The conflation of performance into a percentage mark for each assessment item results in the descriptive power of the assessment being lost (Martin, 1985) since there is no separate reporting of performance on different aspects of the task, and poor performance in some aspects of the task can be compensated for by better performance elsewhere. However the percentage marks are useful to indicate the general standard of work produced by individual pupils and by the class as a whole. That this standard is at least satisfactory in all instances (using 50% as a benchmark) and high overall (the average mark gained was 80.2%) arises in part from the instructional design, where a range of measures was in place to ensure that pupils possessed the prerequisites for the next topic. This approach is in keeping with Bloom's (1979) ideas on mastery learning. (The detailed discussion on individual items of evidence focuses mostly on pupils' errors in an attempt to explain them, and therefore creates a more negative impression.)

Although there are variations in the quality of performances, the greatest variations are in the extent of pupils' progress through the course. This is reflected in the final grades. Michael's 100% in the topic 8 assessment is in no way equivalent to Chris's 97% in the topic 17 assessment. (However this bears out the earlier finding that slow progress does not equate with poor performance.) The topic 17 assessment is more complex than the topic 8 assessment, and it requires a wider knowledge of programming concepts and techniques to accomplish it. Nevertheless, Michael's achievement in the topic 8 assessment is worthwhile, since through it he demonstrates an understanding of important programming techniques and principles (such as using numeric variables, readability and user-friendliness), and the ability to define a problem which is amenable to solution using the programming techniques he has learned, and to design, implement and evaluate the solution.

6.4.3 *Factors affecting pupils' performances, and the importance of active learning*

The placing and timing of assessments – interspersed with ordinary coursework – has an effect on how well pupils performed. The assessments do

not measure a carefully rehearsed performance; rather they capture pupils in the midst of learning, as they attempt to construct robust understandings and to apply their new found knowledge to obtain solutions to problems.

This is most evident in some pupils' attempts to produce a top-down design, particularly in situations where they must synthesize various elements (newly introduced and/or from previous solutions) in order to produce it. For example, in the topic 14 assessment, four of the six pupils – David, Claudia, Catherine and Kelly – had difficulty with indicating the control structures clearly. Also Dawn's and Carol-Anne's designs contained several inaccuracies, Chris's design was monolithic rather than showing stepwise refinement, and David seemed to disregard his design when he came to write the program: might they each have arrived at a solution more quickly and naturally via a different route? Perhaps their programming styles fit the description of the *briçoleur* (Papert, 1994)? Turtlegraphics is a very visual medium in which the effects of experimentation can be readily appraised, however it is much harder to experiment when most of the operations of a program are hidden, as occurs, for example, in the topic 14 assessment when the program generates a number at random or updates the counter for the number of guesses that the player has taken. Another possible explanation for the difficulties that some pupils encountered with designing their programs is offered by Soloway (1986), where floundering and decomposing the problem in inappropriate ways are consequences of the pupil's failure to make adequate recourse to previously solved problems; in other words, it is a problem of transfer.

Evaluation sometimes proved difficult; some opinions were given without justification and sometimes it appeared that the evaluation question was not understood (see the discussion on the topic 14 assessment). Focus sessions conducted with the full class could provide opportunities for evaluation issues to be aired.

Occasionally pupils did not produce 'tidy' solutions. Although they discovered and corrected errors when they were testing programs at the computer (and nearly all errors in the designs were picked up upon and corrected) they often did not retrace their steps to make corrections to the written documentation. Other 'loose-ends' occurred when Scott and Claire failed to make the editing changes to their user documentation in the topic 13 assessment after testing it with a 'real' user, or when Claudia and others deviated from their screen formats in the topic 14 assessment. Being *meticulous* about checking the work in progress and the finished work, and making any necessary revisions, was important to achieve a successful outcome in the tasks, and this factor

accounted for some of the differences in the marks awarded. (This finding is consistent with earlier findings on certain pupils' failure to monitor, check and self-test when working on topic 9 and 10 problems.) These kinds of inaccuracies may have arisen because pupils had lost interest in the problem by the time they finally got their programs to work, and they were therefore disinclined to check over the solution, preferring to move on to fresh challenges.

On a more positive note, another finding which is consistent with previous findings from topics 9 and 10 is that all of the finished programs ran without generating any error messages. Only Carol-Anne's and David's programs contained a logical error. There were some deficiencies in the formatting of output to the screen, e.g. in Anthony's topic 11 assessment. Pupils' programs were, however, correctly sequenced, modular, designed to be user friendly, and readable for the most part (any faults lay with the internal commentary which may have been omitted at the start of the program, or which may not have described accurately the functions performed by the different parts of the program). The user-documentation which pupils produced for the topic 13 assessment was of a high standard, particularly Carol-Anne's. It is clear that she devoted considerable effort to producing it, and it is likely that a novice user could follow her instructions without difficulty. The opportunity to test the program 'for real', and the requirement to word process the user documentation may have provided the impetus for pupils to produce very polished work. Their inclusion of explanations against the list of instructions to run the program seems to have transferred from their programming activity where internal commentary serves the same purpose (some pupils have used a very similar syntax, e.g. placing the explanation after the instruction separated by a single slash, or grouping instructions together into modules which are related to one task).

Pupils' ability and disposition to engage in *higher order thinking* affected the outcomes: Dawn's attempt to tailor an existing design (topic 11 assessment) failed because she did not *analyse* the similarities and differences between the two problems carefully enough; Bryan, Scott and Bernadette failed to *define* the problem fully before embarking on the solution (topic 13 task 9); David and Kathleen failed to *generalize* from previous solutions when they were unable to set up a counting mechanism correctly in their designs (topic 14 task 12); David failed to *reason* well when he gave as an advantage of the 'next_screen' procedure that, 'It instantly clears the... screen.' (topic 14 task 13); and Kelly, Claudia, Catherine, Kirsty and Lorraine failed to *justify* their opinions (topic 14 and topic 16 assessments). It is evident from these examples that learning to program does present ample opportunities to develop higher order thinking

skills, skills which come into play at all different stages of the solution process. The evidence signals the need for some pupils to become more *rigorous* in their thinking, and for them to learn to draw more effectively on their previous learning experiences in the way that others in the class, like Paula and Caroline, are clearly able to do.

Fragile knowledge will have affected pupils' performances, for example, David may not have recognized the relevant language features or the correct syntax to introduce in order to translate the written design for the topic 14 assessment into a program. Overall, however, there were relatively few instances that one could detect from the assessment evidence where inert knowledge was the most obvious cause of difficulties. Had a different research methodology been used based upon observation and verbal protocols as a pupil, or pupils together, worked a problem, more instances of inert knowledge would have been uncovered (e.g. when one pupil asked another for help or information).

However pupils' thinking skills cannot be considered in isolation from their knowledge. Nisbet (1990, p.2) observes, '...thinking is always thinking about something.' When Resnick and Klopfer (1989, p.6) present their vision of the *thinking curriculum,* they emphasize:

> There is no choice to be made between a content emphasis and a thinking-skills emphasis. No depth in either is possible without the other. The Thinking Curriculum joins content and skill so intimately that both are everywhere.

They describe (p.5) how students must go about their learning if they are to be able to use their knowledge generatively:

> Before knowledge becomes truly generative – knowledge that can be used to interpret new situations, to solve problems, to think and reason, and to learn – students must elaborate and question what they are told, examine the new information in relation to other information, and build new knowledge structures.

Within this study, all of these ways of utilizing knowledge – to interpret new situations, to solve problems, to think and reason, and to learn – are given prominence. According to Resnick and Klopfer, pupils will need to be very active learners if they are to succeed in building a generative knowledge base. The learning environment places upon pupils the responsibility to actively manage their learning. The questionnaire, interview and review task findings which are reported in the next two chapters will demonstrate clearly pupils' attempts to manage their learning actively.

6.5 CONCLUSIONS

The main conclusions arising from the detailed analysis of assessment evidence are indicated below.

1. All pupils performed well in coursework assessments. All of the finished programs ran without generating error messages and there were few logical errors. Pupils' programs were correctly sequenced, modular, user friendly and mostly readable. Through these performances pupils have demonstrated their understanding of programming concepts and principles and their ability to solve programming problems.

2. Being meticulous about checking work in progress and any finished work and making the necessary adjustments and corrections was a factor which led to some pupils performing better than others. However the majority of pupils did pick up on errors (e.g. nearly all design errors were corrected) and achieved successful outcomes without relying on the teacher to check their work. This provides evidence that they have engaged in self-monitoring and self-regulation.

3. Higher order thinking skills came into play at all different stages of the solution process. Pupils' ability to, for example, analyse, compare and contrast, define, generalize, justify, and reason affected the outcomes. The majority of pupils were able to demonstrate these skills in context and to use them in an integrated way.

4. Using portfolio evidence ensures that a wide range of competences can be validly assessed and allows all pupils to demonstrate their achievements, but the process of assessing pupils' performances across a range of different tasks (including the formulation of detailed assessment criteria specific to each task) has been painstaking and time-consuming, as it must be for every teacher who sets out to do it properly. However, when one considers Resnick's (1987) criticisms of testing practices in American education, the benefits of this mode of assessment must be seen to far outweigh its drawbacks. However some of the restrictions imposed by the SEB summative assessment requirements make it desirable to position summative assessment opportunities at the end of the course and to use the assessment portfolio as a purely formative measure.

The assessment evidence also points to two interesting questions which will be returned to in later chapters:

5. Designing solutions presented the greatest challenge for some pupils. Is this because the teaching method led them to adopt a top-down approach which did not 'connect' with their natural ways of engaging with problems (as Papert suggests)? Or are there other explanations, for example, poor analysis of the problem, failure to transfer, or the inherent complexity of the process?

6. Does the assessment evidence demonstrate that pupils are any better at thinking well or solving problems in general, than before? This is a crucial question. On the basis of the assessment evidence alone, this has not been convincingly demonstrated. However consideration of the wider evidence may lead to a different conclusion. Pupils were clearly able to learn and apply a range of thinking and problem solving skills successfully in this particular context, which is far from being straightforward.

	Unit 1	Unit 2	Unit 3
Bryan	x	x	
Stephen	x		
Claire	x	x	
Scott	x	x	
Paula	x	x	x
Claudia	x	x	x
Carol-Anne	x	x	
Dawn	x	x	
Chris	x	x	x
Caroline	x	x	x
Anthony	x	x	
Michael	x	x	
Kirsty	x	x	x
Bernadette	x	x	
Kelly	x	x	x
Catherine	x	x	x
Kathleen	x	x	x
David	x	x	x
Gerard	x	x	
Lorraine	x	x	x
TOTAL	20	19	10

Table 12: The available questionnaire data for analysis.

7 CASE STUDY FINDINGS III: AFFECTIVE RESPONSES TO LEARNING TO PROGRAM

7.1 CHAPTER OVERVIEW

This chapter and the following chapter continue to report on the findings for the case study class, but focusing primarily upon those occasions when pupils were engaged in reviewing their learning, either individually or as a member of a small group. Such occasions were: when completing the questionnaire at the end of each unit; when doing the review tasks; and when being interviewed by me. Pupils also reviewed their learning when using the summary sheets and revision questions at home, and they were asked to comment about how useful it was to do this in the unit 2 questionnaire and during interviews.

In this chapter, the focus is upon pupils' affective responses to learning to program and to the learning environment, as revealed through questionnaire and interview responses. The main section of this chapter (section 7.2) reports on and discusses the questionnaire findings, supplemented by relevant interview findings where the questions that I asked pupils were designed to probe their questionnaire responses. Some of the findings have already been mentioned briefly in the two previous chapters where it was relevant to introduce them. The final section (section 7.3) reviews this evidence and relates it to the evidence on progress and attainment.

In the next chapter, the focus is upon pupils' insights on learning and problem solving, as revealed through review tasks and interview responses. Linked into this discussion is a brief presentation on pupils' performances in practical problem solving using a spreadsheet to enable near transfer from programming to be investigated. This will complete the presentation of findings.

7.2 EVIDENCE FROM PUPIL QUESTIONNAIRES AND RELATED INTERVIEW FINDINGS

7.2.1 Method of analysis

Table 12 opposite indicates which questionnaire data is available for analysis; in every case the pupil had completed the relevant unit. All pupils who had completed a unit filled in a questionnaire, so there is no missing evidence. For units 1, 2 and 3 respectively, 40%, 37% and 20% of questionnaires were filled in

by boys. Overall, the figure is 35% (17 out of 49 questionnaires). This gives the girls a slightly stronger voice.

The distinct areas of focus will be:

- overall response rates to items (see 7.2.2);
- pupils' prior experiences of programming (see 7.2.3);
- how interesting pupils found the work to be in each unit (see 7.2.4);
- the programs pupils liked, with reasons (see 7.2.5);
- dislikes (see 7.2.6);
- self-assessments of understanding (see 7.2.7);
- self-assessments of confidence (see 7.2.8);
- pupils' appraisals of the resources (see 7.2.9);
- preferences for working on their own or with a partner, and for working at their own pace (see 7.2.10);
- satisfaction with progress (see 7.2.11); and
- were pupils looking forward to doing the next unit? (see 7.2.12).

The precise wording of items is indicated in each sub-section. A quantitative analysis of questionnaire responses can be found in appendix 3a. This indicates frequencies and percentages for each item and includes a separate breakdown by gender and also by attainment in programming coursework. Percentages have been used to aid comparisons, but it should be borne in mind when interpreting the figures that the number of respondents to each item is small. There is discussion of gender and attainment differences in the data within the sub-sections below.

Performing an analysis of the questionnaire data on the basis of attainment in programming coursework (according to whether the pupil gained a credit or general grade) enables trends to emerge which are otherwise obscured. Because some pupils drop out progressively from the statistics, it can be difficult to determine from totals and percentages for the class as a whole, whether the overall responses are consistent from one unit to the next. Since there is a one-

PRIOR EXPERIENCE

UNIT 1 (QU. 12)

Had you ever done any computer programming before doing Unit 1?

yes	4	20%
no	16	80%

If yes, what had you done?

Figure 25: Questionnaire responses (unit 1, qu. 12 – prior experience).

INTEREST

UNIT 1 (QU. 1)
How interesting did you find the work?

very interesting	3	15%
quite interesting	16	80%
quite boring	0	0%
very boring	1	5%
	N = 20	

UNIT 2 (QU. 1)
How interesting did you find the work in Unit 2 compared with Unit 1?

more interesting than Unit 1	10	53%
about the same	8	42%
less interesting than Unit 1	1	5%
	N = 19	

UNIT 3 (QU. 1)
How interesting did you find the work in Unit 3 compared with Unit 2?

more interesting than Unit 2	6	60%
about the same	4	40%
less interesting than Unit 2	0	0%
	N = 10	

Figure 26: Questionnaire responses (units 1–3, qu. 1 – interest).

to-one correspondence between those pupils gaining a credit grade and respondents to the unit 3 questionnaire, then using the programming coursework grade provides a straightforward basis for analysing the data more carefully.

A statistical analysis has been performed on the responses to all closed items using a Chi-squared comparison. Any statistically significant results (at 5%, 1% or 0.1%, using a two-tailed test) are highlighted below. Because of the small sample size and the use of a non-parametric test, not many results achieved statistical significance. The categories which are used to describe the responses to open items emerged from my analysis of the data. The class teacher performed a reliability check on my classifications of the data, using a sample of the evidence drawn from pupils' responses to the unit 1 questionnaire. There was a 90% agreement with my classifications, which I considered to be acceptable. The manner in which the findings are presented below combines quantitative and qualitative methods of analysis.

7.2.2 *Overall response rates to items*

There was a very high response rate to individual items including open response items (approximately 98% overall), and less than 1% of responses were classified as invalid. The girls responded more fully to open items (i.e. their responses were longer and contained more substantive points) in all three questionnaires – 72% of the total valid classifications (and excluding categories such as, 'no reason stated') were from girls.

The profile of pupils' individual responses to closed items is presented in appendix 3b; the order of presentation corresponds to the order below.

7.2.3 *Pupils' prior experiences of programming*

Only four pupils had any prior experiences of programming (figure 25). Claire and Kathleen had home computers and had typed in some BASIC programs, Michael had programmed games in primary school, and David supplies no information (perhaps he attached little significance to it).

7.2.4 *How interesting pupils found the work to be in each unit*

The indicators on interest are mostly positive (figure 26). For the first unit, only one pupil (Stephen) rated the work as 'very boring', and everyone else

'Which programs did you like the best? Say why you liked them.'				
Categories	Unit 1	Unit 2	Unit 3	Total
A – you can think independently and use your imagination	11	5	0	16
B – you can experience a sense of achievement or satisfaction	5	3	6	14
C – you can learn and practise new techniques	7	3	2	12
D – you can produce useful, realistic and relevant programs	0	7	1	8
E – the problem can be done easily/is easily understood	2	2	2	6
F – the problem is interesting/enjoyable/fun /funny	2	1	2	5
G – the problem is difficult/challenging	1	1	0	2
H – no reasons given for stated preference	0	1	0	1
Total respondents	19	19	10	

Table 13: Questionnaire responses (Units 1–3, qu. 2 – reasons for liking particular programs).

rated the work as 'quite interesting' (the majority view[32]) or 'very interesting'. As pupils progressed to units 2 and 3, just over half of respondents rated the work as 'more interesting' than in the previous unit, with the remainder finding it to be 'about the same', apart from Michael (he rated unit 1 as 'quite interesting', but unit 2 less so). The pupils who rated the course most highly on interest were Paula and Caroline, on each occasion selecting the 'very interesting' (unit 1) or 'more interesting' (units 2 and 3) category.

The girls tended to rate the course more highly on interest than the boys. All three of the 'very interesting' responses in the unit 1 questionnaire were from girls (Paula, Caroline and Lorraine). When the results are combined for units 2 and 3, two-thirds of the responses from girls were in the 'more interesting' category, in comparison to only one-third of the boys' responses.

7.2.5 The programs that pupils liked, with reasons

The findings relate to the first open response item (question 2) in which pupils were asked, 'Which programs did you like the best? Say why you liked them.' The most common reasons that pupils gave for liking particular programs were that the problems presented opportunities to:

A think independently and use your imagination;

B experience a sense of achievement and satisfaction;

C learn and practise new techniques;

D produce useful, realistic and relevant programs.

Table 13 summarizes the findings, including the other categories of response (some responses were categorized under several headings).

The pupil comments (figure 27 overleaf) have been selected to illustrate each of the main themes. The theme of thinking independently and using your imagination (category A) arose most commonly in relation to unit 1 problems ('robot', 'ships', 'dennis', 'joke'). Nearly all of the problems in unit 1 involved the design of output to the screen or to the printer, in addition to the creation of the program itself. The responses to the first review task (which are reported fully in 9.2) also reflect this theme, for example, Paula's review of 'ships':

---

32 I shall use the term 'majority' when reporting the findings if over *two-thirds* of respondents opt for a particular choice, or their responses fall into a particular category.

Thinking independently and using your imagination:

Bernadette *I liked the prize-draw tickets program [T10] because I could use my own ideas and exercise my imagination.*

Caroline *I liked the robot and ship programs [T5] because I designed the robot, ship and the whole program myself.*

Learning and practising new techniques:

Kathleen *The games which involved the coins or the dice [T14]. Because it was interesting to see how the computer counts up throws and stuff.*

Carol-Anne *Poster [T9] – it gave me a lot of practise with using variables. Card [T10] – [it] helped me to tell the difference between string and numeric variables.*

Claudia *I liked Topic 8 best because you had to make up user friendly programs – I feel this is good because people shouldn't have to be brilliant at the computer to be able to work a program.*

Experiencing a sense of achievement and satisfaction:

Gerard *Dennis [T4], because at the end you got a result which seemed as if you had achieved something.*

Catherine *... stars [T12] because it was good to see them in patterns when you had finished.*

Producing useful, realistic, relevant programs:

Bryan *... I enjoyed Topic 7 Task 1 [survey on favourite TV programme] most of all as it allowed you to communicate with the rest of the class.*

Claudia: *[I liked] the games which involved making guesses eg. dice toss, because you could actually use it once you'd finished ...*

Interesting, fun, funny, enjoyable:

Kelly *...I liked the game program [T14] because it was interesting as well as fun.*

Easy, easily understood:

Anthony *Card [T10] – I enjoyed writing the program because it has a repeated process all the way down so it was quite easy.*

Difficult, challenging:

Caroline *I liked "Poster" [T9] and "Card" [T10] because they were a bit more difficult and took a bit more work. They were challenging.*

Figure 27: Questionnaire responses (units 1–3, qu. 2 – the programs that pupils liked, with reasons).

'Ships' [Topic 5]

For this program I had to design a ship on my own, design and write a program and a procedure on my own.

~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

I liked this program because I got to do everything on my own with almost no help. They gave me a design to work from and I worked from that.

~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

I learned most from this program because I got to do it on my own. My own ship design, program, procedure etc.

In unit 2, the authentic nature of the programs (category D) appealed to some pupils – you could process real data (and gather it from other pupils in the class), solve real problems, and design programs with a real user in mind. As Chris put it, it was, 'useful stuff.'[33]

The theme of experiencing a sense of achievement and satisfaction (category B) arose most often in relation to unit 3 problems, the majority of which were fairly substantial and required a lot of effort to do.

These shifts of theme, as well as reflecting the types of problems that pupils encountered within each unit, may also reflect pupils' emerging awareness of their expertise, since, as they have progressed through the course, the range and complexity of the problems they have solved successfully has increased substantially. Also the responses to unit 3 are all from pupils who achieved a credit grade, for whom experiencing a sense of achievement and satisfaction may be very important. As successful learners, they may have a deep, achieving approach to learning (Biggs, 1988).

When one analyses the responses by gender across all three units, different patterns emerge. The girls were more likely, on average, to respond in category C ('learn and practice new techniques' – 31% of girls vs. 12% of boys). The boys were more likely, on average, to respond in category D ('you can produce useful, realistic...' – 25% of boys vs. 12% of girls) or category E ('the problem can be done easily...' – 19% of boys vs. 9% of girls).

---

33 Interview 6, Chris and Lorraine, 22/2/94, Transcripts, p.42.

'Were there any things that you didn't like about the work?'				
Categories	**Unit 1**	**Unit 2**	**Unit 3**	**Total**
A – laborious/time-consuming	6	6	3	15
B – 'no' or a positive comment	6	6	2	14
C – too repetitive	2	3	5	10
D – too difficult/confusing	2	1	1	4
E – an aspect of the learning environment	2	0	0	2
F – particular programs or techniques	0	1	0	1
Total respondents	18	17	9	

Table 14: Questionnaire responses (Units 1–3, qu. 3 – things that pupils didn't like about the work in each unit).

7.2.6 *Dislikes*

The three most frequent responses to the question, 'Were there any things that you didn't like about the work?', were:

A some aspects of the work were laborious or time-consuming;

B 'No' or a positive comment;

C some aspects of the work were too repetitive (this was particularly in response to unit 3).

Table 14 summarizes the findings, including each of the other categories of response. Kirsty's comment in her unit 1 questionnaire (see below) is an example of a positive response to the question:

> ***Kirsty:*** *I liked everything because I felt I had achieved something even if it was quite basic compared to other things done on computers.*

Girls were more likely to respond in category B. Across all three units, the average response rate of girls in Category B was 38%, and that of boys, 20%.

The kinds of things that were mentioned as laborious or time-consuming (category A) were: keying in or editing long programs; writing out programs (e.g. 'I'd much rather go straight onto the computer.', or '...there was a program we had to write and it was very long and took ages to do.'); writing in general (e.g. 'There was a lot more writing [in unit 3] than the last two units.'); the time taken to load programs (pupils had to log on to their user areas at the start of a session on the computer, which might occur several times during a lesson since they were sharing access to computers); the time taken to debug ('...some of it went on forever as I had to find out problems and fix them...'); and even the time to think up a good idea ('You had to use your own ideas and it took me a while to think up a good idea.').

Responses in category A were from a broad cross-section of pupils – girls and boys, and of varying attainments – for example, Paula complained about long programs or the amount of writing in relation to all three units, as did Michael for unit 1. However overall there was a higher response rate from girls in this category (79% of the responses were from girls).

Certain of the complaints about laboriousness may have arisen from the nature of the programming environment, since Acornsoft COMAL lacks some of the facilities available in newer programming environments which are

designed to ease the programmer's load, such as on-line help and a good range of editing facilities. Also poor keyboarding skills may have contributed to some pupils' sense of frustration. In terms of the amount of writing involved, some pupils may have held the expectation that programming is a practical activity, and they may not have expected to do much writing on the course.

Although having to spend a long time on some tasks was a disliked aspect of the work for some pupils, the experience was not always perceived by them as entirely negative, as Kirsty's comment below illustrates:

> ***Kirsty:*** *I didn't like the rainfall programs [in Topic 13] because they took a long time to do but in the end I was very sure about that work.*

Since Paula had complained about, '...when we had to use or write long programs,' in unit 1, one of the questions that I asked her during interview was, 'Do you think you would have got on better if you didn't have to write the program first, before you tried to enter it?'. She replied:

> **P:** *I thought it was better to write the program out first, so that when you are putting it on the computer, you didn't miss out any commands like 'mode 1' or 'clear the screen'. You didn't miss out any commands that you needed to put in – you copied it down from the paper.*

[Interview 3, Paula, 1/11/93, Transcripts p.18–9]

Also, she found that having the program written out in her answer booklet made things a lot easier when it came to entering it.

> **P:** *Well, when you are writing out the program, it was easier to write it in the book first, then onto the computer so that you know what you're doing. You weren't looking around all over the place to write something in or to look at something – it was there in front of you all the time.*

[Interview 3, Paula, 1/11/93, Transcripts p.19]

The main (discernible) reason for pupils finding that some of the work was too repetitive (category C) was that certain problems were viewed as too similar to other problems (e.g. 'You sometimes did the same thing e.g. vans, ship, robot [in topic 5]'). Some comments were less specific, e.g. of unit 3, 'Sometimes it seemed to repeat itself.' Whether pupils considered that the contexts of the problems were too similar within topics, or certain techniques were over-rehearsed, is unclear from the responses – perhaps it was a combination of both. This criticism was made by a cross-section of pupils, including Chris, Lorraine, David, Bryan, Michael and Gerard, but more frequently by boys (who account for 60% of the responses in category C). Criticisms about laboriousness and repetition must, however, be placed alongside pupils' general appraisals of

how interesting they found the work to be (responses to question 1), where overall they expressed less dissatisfaction.

When it comes to the difficulty of the work (category D), Stephen was the only pupil to complain that he couldn't do any of the work in a particular unit ('Were there any things that you didn't like about the work [in unit 1]'? 'Yes. I can't do it.'). Caroline thought that procedures were not explained in enough detail in unit 1, and she did not understand hand tracing and computer tracing in unit 3. Anthony wrote that he didn't like the 'address cards' problem because he kept forgetting to put in the '$' sign. In her unit 2 questionnaire, Kathleen wrote that she didn't like using string variables (category F).

The word 'boring', where it occurred, was associated in pupils' comments with any of the following situations – when tasks were considered by pupils to be laborious or time-consuming, repetitive, or too difficult to understand, e.g.

> *Some of the tasks took too long and got boring. There was a lot more writing than the last two units. [Lorraine's comment about unit 3]*
>
> *It was boring after a while because the whole unit was like three tasks repeated e.g.. coin, dice etc. [Claudia's comment about unit 3]*
>
> *I disliked the hand tracing and computer tracing. I did not understand this and found it boring to read. [Caroline's comment about unit 3]*

I discussed with Bryan which aspects of the work in unit 2 he experienced as too similar.

B: Most of it was just based on the same thing.

MK: Give me an example, then, if you can.

B: The poster and the address cards [Topics 9 and 10] were just the same, but a different message on it.

MK: So, a different message, different output on screen?

B: Yes.

MK: But?

B: Just the same program.

MK: Were the designs [for the programs] also similar?

B: Yes.

MK: You felt you already knew how to do it?

B: Yes.

MK: *I noticed, when I looked at your answer booklet, that you actually made* more *errors in the second one than the first – why do you think that was?*

B: I don't know. SC: Boredom!

MK: *The impression I had was that you hadn't referred back to your previous solution [which pupils were prompted to do in the task outline]. Were you looking back?*

B: No, not really...

[Interview 10, Carol-Anne, Bryan and Scott, 25/5/94, Transcripts p.95–6]

Lorraine had also identified the same set of problems as being too repetitive.

L: ...just now and again like in the variables and stuff, you were repeating in different programs and different tasks, you were repeating the thing over again. I can't remember exactly what ones it was but there was a couple towards the end, like, when you were doing string variables.

MK: *And you thought it was more repetition than you needed to understand it?*

L: Yes.

[Interview 6, Chris and Lorraine, 22/2/94, Transcripts p.44]

The difference here is that Lorraine's work improved from the first task to the second (she made no errors at all in 'address cards'), whereas Bryan's work deteriorated.

One can see, from the discussion with Bryan, how a vicious circle can arise in situations where the quality of work produced by the pupil is erratic. The longer that is spent on trying to solve a problem, the more tedious, and perhaps frustrating, the work becomes, the more careless errors are introduced, thus slowing progress even further. Since Bryan reckoned that the second problem was almost identical to the first, then naturally he wasn't keen to do it, and he didn't accomplish it well. The evidence suggests to the teacher, at least on the surface, that he needs even more practice on similar types of problems. Over time, the pupil experiences less variety than other pupils who have made better progress and who have thus had the opportunity to tackle a wider range of problems (and as pupils gain in knowledge and competence, there is greater scope for solving a variety of problem types). Also it may be harder for the pupil to perceive the *relevance* of the work, since he or she has not progressed to the stage of solving complex problems which demonstrate more effectively the power and versatility of the programming language as a tool for solving 'real world' problems. There is evidence of this in Bryan's response to the first review task, when he writes about the 'currency conversion' problem in topic 11:

EXPLANATIONS

UNIT 1 (QU. 4)
Were you able to understand what was being taught...?

all of the time	5	25%
most of the time	12	60%
some of the time	2	10%
hardly any of the time	1	5%
	N = 20	

UNIT 2 (QU. 4)
Were you able to understand what was being taught...?

all of the time	4	21%
most of the time	10	53%
some of the time	5	26%
hardly any of the time	0	0%
	N = 19	

UNIT 3 (QU. 4)
Were you able to understand what was being taught...?

all of the time	2	20%
most of the time	8	80%
some of the time	0	0%
hardly any of the time	0	0%
	N = 10	

WRITTEN INSTRUCTIONS

UNIT 1 (QU. 5)
Were the instructions in the Task Sheets easy or difficult to follow?

very easy	5	25%
quite easy	14	70%
quite difficult	0	0%
very difficult	1	5%
	N = 20	

UNIT 2 (QU. 5)
Were the instructions in the Task Sheets easy or difficult to follow?

very easy	4	21%
quite easy	12	63%
quite difficult	3	16%
very difficult	0	0%
	N = 19	

UNIT 3 (QU. 5)
Were the instructions in the Task Sheets easy or difficult to follow?

very easy	0	0%
quite easy	8	80%
quite difficult	2	20%
very difficult	0	0%
	N= 10	

Figure 28: Questionnaire responses (units 1–3, qus. 4, 5 – understanding).

CURRENCY CONVERSION TABLE [TOPIC 11]

In this task I set up a currency conversion table. I had to copy [enter] the main program, get a printed listing, [also code and enter each procedure], run the program adding [inputting] an appropriate nation and currency and get a sample run.

I liked the program because it was quite positive i.e.. it could be used for something, unlike a program for a robot or a series of squares.

From this program I learned that COMAL could be used for something useful.

It has taken Bryan a considerable time to appreciate that what he has been learning could be useful.

Dawn and David were critical about aspects of the learning environment (category E):

Dawn: *I think the work would have been better if you worked with a partner because both people would always be on a computer also if you needed help you could ask your partner instead of the teacher all the time.*

David: *I didn't like the questions being in 1 book and the answers in another.*

Dawn reiterated her criticism later in the unit 1 questionnaire when she was asked whether she preferred to work on her own or with a partner (question 9), where she added that, '...you could do the work twice as fast,' because a computer would always be available.

David expanded on his point when he gave his reason for not liking the answer booklet in his response to question 7, 'I didn't like the answer booklet because the questions were in another book so you couldn't really take it home.'

Therefore they both felt that their progress was being hampered by the way in which the resources were used in the classroom.

7.2.7 Self-assessments of understanding

The majority of respondents were confident about their understanding throughout the course (figure 28). In unit 1, 85% of respondents indicated that they were able to understand what was being taught, 'all of the time', or, 'most of the time', and the corresponding figures for units 2 and 3 were 74% and 100% respectively. Paula was the only one to indicate that she understood, 'all of the time', for all three units. Those pupils who indicated that they understood only, 'some of the time', were Bernadette and Gerard (units 1 and 2), and Claire, Scott and Dawn (unit 2 only). None of this group finished unit 3.

Stephen indicated that he understood the work in unit 1, 'hardly any of the time' (which ties in with his response to Qu. 3, where the reason he gave for not liking the work was that he couldn't do it). He was the only pupil who did not manage to complete unit 2[34].

Also the majority of respondents were able to follow the instructions in the task sheets without difficulty. 95% of respondents indicated that they found the instructions in the task sheets for unit 1, 'very easy', or, 'quite easy', and the corresponding figures for units 2 and 3 were 84% and 80% respectively. The pupils who reported that the instructions were, 'quite difficult', were Bernadette, David and Gerard (unit 2 only), and Claudia and Kathleen (unit 3 only). Stephen was the only pupil to report that he found the instructions, 'very difficult' (unit 1).

When the results are analysed by gender across all three units, the girls were, on average, more than twice as likely to indicate that they could understand what was being taught, 'all of the time' (28% vs. 13%), and they were more than four times as likely to indicate that they could follow the instructions very easily (25% vs. 6%).

Pupils' reading comprehension skills could impact considerably on their progress in a course of this nature where they are dependent upon reading as their primary source of information. The class teacher had no prior knowledge of pupils' reading comprehension skills when they joined her class, although she would have been informed if any pupil had been given specialist learning support in S1 or S2.

During interviews, I explored with pupils the *nature* of their difficulties with understanding, and how they attempted to cope with these difficulties.

Many pupils asked for help, from the class teacher or another pupil, if they felt that they didn't understand. Chris also tried to use the help hints if he couldn't get his programs to work:

CH: I'd ask the teacher or sometimes, I went once to the help things [help hints – debugging aid], but it didn't have it in it, so I didn't use it.

MK: Did you ever ask any of the other children in the class?

---

34 When the results from Qu. 4 (unit 2 questionnaire) are analysed by attainment (according to whether pupils gained a credit or general grade) the outcome is statistically significant at 1%, using a two-tailed Chi-squared comparison.

CH: Uh-huh. Michael xxxx, the boy who sits beside me.

MK: Was he able to help you sometimes?

CH: Most of the time – he was usually ahead of me.

[Interview 1, Chris, 15/10/93, Transcripts p.8]

Lorraine would make a further attempt to understand before she sought help:

L: I looked at it again and tried to understand it more – if I couldn't I asked for help.

MK: Who did you ask?

L: Mrs xxxx.

MK: Just Mrs xxxx?

L: Sometimes I'd ask Paula because Paula sits beside me.

MK: Was she able to help sometimes?

L: Yes.

MK: Were you able to help her sometimes?

L: Yes.

[Interview 2, Lorraine, 15/10/93, Transcripts p.16]

Scott wasn't able to understand how to change colours on one line when he was working on 'dennis' in topic 4. He used trial and error, as well as seeking help from the teacher:

MK: Eventually you understood it?

SC: Yes.

MK: How did you get to understand it?

SC: Just the teacher's help and trial and error.

MK: Do you think it could have been better explained?

SC: Not really because it was kind of hard anyway so....

[Interview 5, Scott, 29/11/93, Transcripts p.33]

Sometimes a new concept or technique was difficult to understand when you first encountered it, as Chris explained:

CH: Well, see when I was doing the variables at first, it was a bit like, I don't know, I didn't understand them at first, but then it started to get easier.

MK: Because there were enough problems about variables, you grasped it eventually?

CH: Yes.

MK: Do you think it could have been presented better?... Or do you feel it was just [hard to understand] because it was new to you?

CH: Just because it was new.

[Interview 6, Chris and Lorraine, 22/2/94, Transcripts p.45–6]

Although David felt that he understood the work in unit 2, 'most of the time', there were occasions when he felt he did not.

MK: Any places where you felt you were having problems?

DA: The only time I had problems was when I didn't read the question properly – [I] didn't read it all.

MK: Did you discover that for yourself, or did it have to be pointed out to you?

DA: After the first time it was pointed out, I managed.

[Interview 11, Dawn and David, 20/6/94, Transcripts p.114]

Not reading the question properly (or failing to analyse a problem carefully enough before embarking on a solution) was quite a common source of difficulties for pupils. It emerged as a likely explanation for some of the difficulties encountered by pupils in topics 9 and 10 and in their coursework assessments, which were discussed in the two previous chapters.

Paula, Kathleen and Caroline were interviewed together when they finished unit 2. Kathleen made an observation that you sometimes *think* that you understand when you first encounter a problem or task, but it is only by working through it that you begin to get a real understanding:

MK: You [all] understood what was being taught, 'all of the time' [according to their questionnaire responses]. Does that mean that you didn't have any problems, then?

ALL: [Laughter!]

MK: Do you think the work was too easy if you understood it all of the time?

ALL: No.

MK: What makes you say, no?

KA: Because it's different reading it [the problem or task] and understanding it, but when you go and do it yourself, it's like you sometimes make mistakes then that you didn't think of when you were reading it.

OTHERS: [nod agreement]

[Interview 7, Paula, Kathleen, Caroline, 9/5/94, Transcripts p.58]

This seems to be a common theme in a number of the responses, and also elsewhere in the data; if a concept or technique is at all complex, then it is important to put your understanding to the test. Therefore a willingness to *persist* in the face of difficulties or uncertainty is an important factor for success both in terms of accomplishing a task and securing understanding. Kathleen's response demonstrates both a realistic expectation and a positive outlook – some unexpected difficulties may emerge as you work on a problem, and therefore you can secure a better understanding by overcoming them, which equates with an incremental conception of ability (Dweck and Elliott, 1983). Kathleen was indeed very persistent in her efforts; a lack of a realistic expectation or positive outlook may explain why some other pupils (such as Bryan) were more easily put off.

7.2.8 Self-assessments of confidence

Pupils were asked in each questionnaire to indicate whether they were, 'confident', or, 'not sure', about a range of techniques that were introduced in the unit. In all, 32 techniques were listed (9 in the unit 1 questionnaire, 11 in the unit 2 questionnaire, and 12 in the unit 3 questionnaire). The list does not cover all of the techniques that were taught, e.g. it omits evaluation skills because it was felt that pupils might find it difficult to assess their confidence in this area without some prior discussion to prompt their thinking. The techniques ranged across *analysis skills* (e.g. identifying the input, process and output for a program), *design skills* (e.g. designing programs by listing the main steps in order), *language features* (e.g. using the PROC and END PROC commands), *standard algorithms* (e.g. to compute a total), *process skills* (e.g. saving and loading programs), *testing and debugging skills* (e.g. doing a hand trace of a program), and *documentation skills* (e.g. how to write programs which are readable). The order in which the techniques are listed in the questionnaires corresponds to the order in which they were introduced. In the unit 3 questionnaire there is an indication of the topic where each technique was first introduced, since it was discovered during the second phase of interviews that most pupils experienced some difficulty with recollecting techniques that were introduced early on in unit 2, and which did not feature prominently in later topics (for the majority of the class, the work in unit 2 spanned over the Christmas and Easter breaks). Indicating the topic against each item made it easier for pupils to refer back to the relevant point in the unit if necessary, before responding.

Item	Technique	C[1] [N]	NS[2] [N]	C [%]
1A	Entering COMAL commands	18	2	90
1B	Entering and editing programs	18	2	90
1C	Listing and running programs	19	1	95
1D	Saving and loading programs	20	0	100
1E	Getting printed output and printed listings	20	0	100
1F	Designing programs by listing the main steps in order	16	4	80
1G	Writing programs using //, PRINT, END commands	19	1	95
1H	Writing programs – COLOUR command	16	4	80
1I	Writing programs – PROC, END PROC commands	17	3	85
2A	Identifying the input, process & output for a program	18	1	95
2B	Using pre-written procedures	16	3	84
2C	Testing a program by entering different data when run	17	2	89
2D	Writing programs which have numeric variables	12	7	63
2E	Comparing expected and actual output	16	3	84
2F	How to write programs which are user friendly	16	3	84
2G	How to write programs which are readable	11	8	58
2H	The difference between a numeric and string variable	13	6	68
2I	Using a FOR loop in a program	11	8	58
2J	Step-wise refinement	14	5	74
2K	Finding and correcting bugs in a program (debugging)	15	4	79
3A	... the solution uses a FOR loop to output a table (T11)	10	0	100
3B	Using TAB to align the headings and columns (T11)	10	0	100
3C	... the solution uses one loop nested inside another (T12)	8	2	80
3D	Using a parameter with a procedure (T12)	4	6	40
3E	Doing a hand trace of a program (T12)	8	2	80
3F	... the solution uses a loop to calculate a total (T13)	10	0	100
3G	as 3F, except the values are read rather than input (T13)	7	3	70
3H	Making up a trace table (T13)	8	2	80
3I	Getting the computer to trace a variable (T13)	4	6	40
3J	... the solution involves counting occurrences (T14)	9	0	100
3K	Using a branch (IF .. THEN ..) (T14)	10	0	100
3L	Using a conditional loop (REPEAT .. UNTIL ..) (T14)	10	0	100

[1] C – Confident [2] NS – Not sure

Table 15: Questionnaire responses (Units 1–3, qu. 6 – confidence about the main aspects taught in each unit).

The results (as frequencies and percentages) are indicated in table 15 opposite. When one sees all of the techniques listed together in this manner, one begins to appreciate what a sizeable task it is for pupils to learn them; the knowledge base for programming is extensive, and very few of the concepts would already be familiar ones.

From this it can be seen that, overall, there is a high level of confidence about the techniques that have been taught (83% of the responses are in this category). Respondents are particularly confident about the techniques in unit 1 and the standard algorithms in unit 3 (the algorithm that pupils were least sure of was nesting of loops, which is not surprising since it is quite complicated to trace the order of execution of the program – it is the equivalent in mathematics of generating truth tables and permutations). Pupils were least confident about variables (2D, 2H), FOR loops (2I) and the readability of programs (2G) in unit 2, and parameters (3D) and getting the computer to trace the value of a variable (3I) in unit 3. However all of the common algorithms in unit 3 utilize variables and most utilize FOR loops, and this indicates that those pupils who completed unit 3 and who were unsure initially became confident about using these language features. Since the use of loops and variables within standard algorithms contextualizes them, pupils had the opportunity in unit 3 to gain concrete experience of how the language features come into play to achieve a variety of common programming goals.

It emerged in the discussions during interviews that pupils were unsure about parameters and getting the computer to trace the value of a variable because the techniques were only encountered once in the unit and pupils tended to forget about them, or they felt (quite sensibly) that they needed more practice at using them before they could feel confident.

There was considerable confusion about what was meant by a readable program and a user friendly program (2F and 2G), often arising from a failure to differentiate between the end-user and the programmer, and consequently between the end-users' and programmers' needs (this finding was presented earlier in 6.3.7 when pupils' evaluation skills were discussed in relation to the topic 14 assessment). The following extract, from Claire's, Bernadette's and Catherine's unit 2 interview, illustrates this. The initial difficulty for Claire and Bernadette was in understanding what user friendly meant.

MK: *Claire and Bernadette, you weren't sure about that [how to write programs which are user friendly]?*

C: Yes.

BE: I didn't really understand what user friendly meant.

MK: So, who is the user, then?

BE/C: Us.

MK: [You are] both very definite about that! Let's say – put it the other way round, and say that you were the programmers. Who would the users be?

BE: People that use it.

MK: So if it's a user friendly program, what do you think that means?

C: People can use it. People know what it's on about.

BE: It's nice to use!

MK: And what kind of features would it have that would make it user friendly? What would you expect to find?

C: Words, instead of symbols. You know how that one that says print.... three to do something.

MK: Sorry, what was that Claire?

C: You know the loop? – says print to do – you say, 'cards do', or something like that. The counter [Claire is referring to a part of the coding].

MK: And you think the user would have difficulty in understanding that?

C: Yes.

MK: Is the user going to see that?

C: No.

MK: So that doesn't matter to the user. So what might matter to the user, then?

C: Make sure the questions are clear that the computer is asking so the user knows exactly what he has to do. [She is now on the right track. Discussion follows on user friendly features.]

MK: So if that's user friendliness, then what's a readable program?

C: [You write it] so that you can read it.

MK: What could you add to the program to make it readable?

CT: They wee lines that do that [gestures – two diagonal slashes] then you write down after what it is... Like 'VDU 2 start printing', and 'stop printing'.

[Interview 9, Catherine, Bernadette and Claire, 24/5/94, Transcripts p.81–3]

% 'Confident' Responses	Unit 1	Unit 2	Unit 3
Pupils gaining a credit programming coursework grade.	N = 10 98%	N = 10 87%	N = 10 82%
Pupils gaining a general programming coursework grade.	N = 10 83%	N = 9 64%	N = 0
All pupils.	N = 20 91%	N = 19 76%	N = 10 82%

Table 16: Questionnaire responses (Units 1–3, qu. 6 – confidence, comparisons by attainment).

Later in the interview Claire demonstrated her new-found understanding of the term 'user friendly' when she suggested that we should rewrite the summary sheets to make them more user friendly (which raised a laugh).

When one examines the responses of individuals, Paula, Catherine, Lorraine and Kathleen emerged as the most confident of those pupils who completed all three units; Paula indicated that she was confident about all but one technique, Catherine and Lorraine all but two, and Kathleen all but three. Carol-Anne and Michael emerged as the most confident of those pupils who completed only two units; they were confident on every count. However, not all pupils in the class were this confident. Dawn in particular went through a crisis of confidence at the stage when she completed unit 2 and she rated herself unsure on 10 out of 11 counts, however her confidence had returned by the end of the course[35]. Some other pupils were not confident about certain techniques encountered in unit 2 – Gerard (unsure on 7 counts), Claire and Bernadette (unsure on 5 counts), Anthony, Kirsty and Kelly (unsure on 4 counts). There is no doubt that the work towards the end of unit 2, from topic 8 onwards where variables and loops are introduced, proved to be a difficult part of the course for many pupils (the evidence on pupils' progress in topics 9 and 10 bears this out).

Stephen is the only pupil who indicated that he was unsure of most of the techniques introduced in unit 1 (on 6 counts out of 9); in contrast, every other pupil in the class was confident about all aspects in unit 1 (12 pupils), all but one aspect (3 pupils) or all but two aspects (4 pupils).

A number of trends emerge in the data when it is analysed for each unit separately, and broken down by attainment and also by gender. Table 16 (opposite) shows the outcome when the data is analysed according to the programming coursework grade awarded to the pupil (the individual confidence ratings are combined into a total). The two clearest trends are that, overall: (i) pupils who gained credit grades were more confident than other pupils, particularly in relation to unit 2 aspects; (ii) for both groups, levels of confidence diminished as pupils progressed to the later units, especially for

---

35 I am basing this finding on her unit 3 questionnaire, which is not included in the analysis since she didn't finish unit 3, but which she filled in for me before her last interview. She has written 'confident' against each one of the items that she had been unsure of previously.

% 'Confident' Responses	Unit 1	Unit 2	Unit 3
Girls.	N = 12 98%	N = 12 77%	N = 8 83%
Boys.	N = 8 79%	N = 7 75%	N = 2 83%
All pupils.	N = 20 91%	N = 19 76%	N = 10 82%

Table 17: Questionnaire responses (Units 1–3, qu. 6 – gender comparison on confidence).

Category	Items	Confident Ratings (Girls) %	Confident Ratings (Boys) %	Difference (Girls over Boys) %
Analysis And Design Skills	1F, 2A, 2J	89%	73%	16%
Language Features	1G, 1H, 1I, 2D, 2H, 2I, 3B, 3D, 3K, 3L	82%	70%	12%
Standard Algorithms	3A, 3C, 3F, 3G, 3J	90%	89%	1%
Process Skills	1A, 1B, 1C, 1D, 1E, 2B	96%	89%	7%
Testing And Debugging Skills	2C, 2E, 2K, 3E, 3H, 3I	78%	78%	0%
Documentation Skills	1E, 2G	83%	90%	-7%

Table 18: Questionnaire responses (Units 1–3, qu. 6 – gender comparison on confidence, by main categories).

Item	Technique	Confident (Unit 1)	Now Confident (Unit 2)	Still Unsure (Unit 2)
1A	Entering COMAL commands	18	18	0
1B	Entering and editing programs	18	19	0
1C	Listing and running programs	19	19	0
1F	Designing programs	16	19	1
1G	//, PRINT, END commands	19	19	0
1H	COLOUR command	16	17	2
1I	PROC, END PROC commands	17	19	0

Table 19: Questionnaire responses (Unit 2, qu. 7 – re-assessments of confidence).

pupils gaining a general award. These trends are masked in the overall figures, but the differences are statistically significant[36].

Table 17 shows the outcome when the data is analysed by gender. The girls were much more confident than boys in relation to unit 1 aspects. Then there is a considerable drop in the girls' confidence (of 21%) which is not mirrored in the statistics for the boys (where there is only a 4% drop), so that the figures almost equate by unit 2. Three of the girls – Dawn, Bernadette and Claire, between them account for much of this drop[37].

When the items from all three questionnaires are grouped under the following headings: – analysis and design skills, language features, standard algorithms, process skills, testing and debugging skills, and documentation skills, it is in the areas of *analysis and design* and *language features* that the greatest differences emerge[38] (table 18).

In the unit 2 and unit 3 questionnaires, pupils were asked to re-assess their confidence about any techniques that they had indicated they were unsure of in the previous questionnaire (this was an open-response item; pupils could choose to make a list or write comments). The results for this item in the unit 2 questionnaire are indicated in table 19 – the first column of figures shows the frequency of 'confident' responses from the unit 1 questionnaire, the second column shows the adjusted figures by the end of unit 2, and the third column shows the number of pupils who had indicated that they were still not sure about a technique. Only Claudia failed to respond in relation to (1A). Note that Stephen has dropped out from the statistics in the second and third columns because he did not finish unit 2. This means that everyone else in the class was confident, by the end of unit 2, about all of the aspects that were taught in unit 1, except Gerard, who was still not sure about (1F) and (1H), and Scott, who was still not sure about (1H). Three responses (see below) did not show up in this

---

36 Each of the following tests proved to be significant using a two-tailed Chi-squared comparison; (i) for unit 1, comparison by attainment (at 0.1%); (ii) for unit 2, comparison by attainment (at 0.1%); (iii) for pupils gaining credit grades, comparison across units (at 1%); (iv) for pupils gaining general grades, comparison across units (at 1%).

37 Each of the following tests proved to be significant using a two-tailed Chi-squared comparison; (i) for unit 1, comparison by gender (at 0.1%); (ii) for girls, comparison across units (at 0.1%).

38 The differences shown in the last column of the table are *not* statistically significant at 5% using a Chi-squared, two-tailed comparison.

analysis because the pupils had indicated earlier, in the unit 1 questionnaire, that they were confident about all of the techniques listed there:

> ***Carol-Anne:*** *I feel even more confident about entering and editing because I got a lot of practice during Unit 2.*
>
> ***Caroline:*** *By the end of Unit 1 I finally understood procedures but I fully understand them now and feel more confident about using them.*
>
> ***Kathleen:*** *I feel quite confident about most things but there were some things that I couldn't do but since then I have picked it up good.*

Carol-Anne's, Caroline's and Kathleen's responses are nevertheless interesting because they signal a self-awareness of their growing competence and understanding.

It is not worthwhile to show the adjustments for (2A)–(2K) resulting from the responses to the unit 3 questionnaire because Kirsty was the only one of six pupils to respond to the question (her responses on four counts were all positive). Interview transcripts reveal a likely explanation for this – that some pupils did not have their unit 2 questionnaires to refer to as they completed the unit 3 questionnaire.

An interesting finding to emerge from discussions during interviews (and an aspect which I probed) was the variety of factors that pupils considered when reaching an assessment of their confidence about the techniques they had encountered.

Not having a clear or full *understanding* was one of the most common reasons that pupils gave for not being confident (Caroline's comment about procedures above, and Claire's and Bernadette's confusion about user-friendliness are two examples of this).

Not recognizing a technique at the time when completing the questionnaire was another reason that pupils gave for not being confident, even if, once reminded of the technique during interview, the pupil recollected having understood it earlier. Claudia's and Bryan's comments below illustrate this perfectly. In this situation the factor that is being considered by the pupil is *durability*.

> ***Claudia:*** *I feel there should be something in between confident and not sure. I think I was [confident] at the time, I remember just doing it [a computer trace] – but if you asked me to do it now, I couldn't, I don't think.*
>
> *[Interview 17, Claudia, Kathleen, 27/1/95, p.216]*

Bryan: *I couldn't remember what a parameter was [when I filled in the questionnaire]...I understood it at the time, but...*

[Interview 16, Scott, Bryan, 20/1/95, Transcripts p.203]

Mastery and *retention* were two other important factors to emerge; if, sometimes, pupils failed to carry out a technique successfully or they forgot about some aspect of the task, then they would usually state that they were not confident about it. Chris made several mentions of such situations, for example, with regard to editing: 'I could do it, but I wasn't exactly always getting it right,' and readable programs: 'I keep on forgetting to put a bit in there [points to listing], like the wee [gestures – the two diagonal lines which are used to mark internal commentary].'[39]

Such problems were often remedied over time as pupils had more opportunities for practice. Such was the case for Kathleen:

KA: It was just like sometimes I forgot to write in 'END PROCs' and I was just writing 'END' all the time and the program wasn't working, so I had to keep going back.

MK: That's something that you now remember to do? [according to Kathleen's Unit 2 questionnaire she is now confident about using procedures]

KA: Yes.

[Interview 7, Paula, Kathleen and Caroline, 9/5/94, Transcripts p.61]

However not all pupils felt that they did get enough opportunities for practice. While Claire would have preferred more practice tasks, Bernadette simply felt that she would have liked to be able to take her time more, which she didn't think she could afford to do if she wanted to keep up with others in the class. Claire had told me that she didn't ever feel confident about her ability to write programs:

MK: When you are writing the program you don't ever feel confident that you are going to be able to do it, whenever you start off?

C: No.

MK: But you do [succeed] eventually, don't you? How do you get there, then, from feeling you can't do it at all, to being able to do it?

C: Ask the teacher.

MK: Do you ask for a lot of help?

---

39 Interview 1, Chris, 15/10/93, Transcripts p.9; Interview 6, Chris and Lorraine, 22/2/94, p.47.

C: Yes.

MK: And you feel by keeping on doing that [asking for help], you are able to do the work, to keep going with it?

C: [nods agreement]

MK: Do you feel it would be better to go back and do some of the earlier work again, or would you rather just keep on with what you are doing?

C: I'd rather keep doing it but do more programs. Do more of them.

MK: What do you mean by, 'do more of them'?

C: I don't know, more practice at it.

MK: Do you think the pace is a bit quick for you, sometimes? You'd rather do more, [and] similar problems?

C: Yes.... one task on one thing then straight onto a different thing.

MK: [Turning to Bernadette] Are you feeling a bit rushed, Bernadette?

BE: Yes, we should have more time to do them, so that you understand it better.

[Interview 9, Catherine, Bernadette and Claire, 24/5/96, Transcripts p.86–7]

Getting the balance right between practice and incremental tasks can be difficult to achieve for individuals. Some intended incremental tasks may not be experienced as incremental by pupils who already possess the relevant knowledge and skills. Also some intended practice tasks may not be experienced as such by pupils whose knowledge and skills are insecure[40]. Claudia suggested that pupils' self-assessments of their confidence could play a part in determining the appropriate 'next steps' for them:

> ***Claudia:*** *I think maybe there could be a part in the questionnaire and in the task sheets that you could say if you are not sure you can do this, but if you are sure you can move on. Because sometimes you were, [it] just depends on how you grasped it.*
>
> *[Interview 17, Claudia and Kathleen, 27/1/95, p.211]*

Being able to carry out a technique *without any prompting or assistance* from the teacher or other pupils was a factor which affected pupils' confidence, as is evident from Paula's and Caroline's comments about the revision questions:

P: ...you were on your own so that you couldn't really say like [to] whoever was sitting next to you, 'What's that?', 'Oh, right, that's it', then just forget about it again. You learned it and you remembered it...

---

40 This is essentially an issue of differentiation. There is evidence from other studies in secondary science and mathematics of a frequent mismatch occuring between the difficulty of tasks allocated to pupils, and their abilities (Simpson and Ure, 1993).

C: *They [the revision questions] were good to help you to understand it because you had to do them at home when you didn't have any of the other books. You really had to think – it really tested you – what you really knew and what you didn't.*

[Interview 7, Paula, Kathleen and Caroline, 9/5/94, p.61]

Some pupils were cautious about the *range of circumstances* in which they could successfully apply a technique – e.g. could they perform it if no hints were given in the notes, or if they couldn't refer back to an earlier example, or if the set problem was not obviously similar to other problems that had been solved previously.

Bryan and Scott had successfully accomplished the tasks in topic 12 which involved using nested FOR loops to output square, rectangle, triangle and parallelogram shapes built from '*' characters. But neither felt confident about the technique.

B: It would be a bit easier to use it now because you've used it [before] and you know what it is. So even if you don't know what it is [when you next encounter it] you can look back at the other programs [in Topic 12]...

MK: Yes, but do you think you would recognise <u>when</u> it was appropriate to use that technique?

B: No.

MK: Scott?

SC: I don't think so.

MK: Do you reckon that you would be more confident to use it if, for example, you were told, 'Use this technique to solve such and such a problem?.'

B: Yes.

SC: It would have been easier if they [the learning materials] told you what to use and when to use it.

[Interview 16, Scott and Bryan, 20/1/95, p.200–1]

Bernadette was uncertain that she could do a hand trace on a program unless she had an earlier example to hand, even although she had no difficulty in describing what it was to me.

BE: I could do it [a hand trace] like looking back and copying the last one, but I don't know if I could do it just straight off.

MK: Could you just describe to me what you think it means by doing a hand trace?

BE: Writing down what the computer does to get the answer.

[Interview 12, Dawn and Bernadette, 18/11/94, Transcripts p.136]

Caroline discussed her approach to designing programs as follows:

> ***Caroline:*** *I really need something to look back, a similar problem to look back at, because I'm not very good at writing them [the designs] just straight from scratch...you go back to a similar one to give you an idea.*
>
> *[Interview 25, Caroline, 20/3/95, Transcripts p.361]*

It is clear that Caroline relies heavily on her ability to transfer the techniques that have worked successfully in the past (in this instance, design templates) to new and unfamiliar situations.

The factors which pupils have identified as affecting their confidence therefore range across all four criteria for skills training that are outlined by Nisbet and Shucksmith (1984), i.e. mastery, retention, durability and transfer (they argue that schools tend to neglect the last of these). In addition, pupils have identified that confidence often only emerges with understanding. Whenever pupils could not bring concepts, principles or skills to bear on new problems or situations it seems that they were *not* confident about their understanding, and this suggests that pupils accord with Gardner's (1993) view of what it means to understand.

The findings indicate that pupils become more confident about particular techniques if they are given ample opportunities to practise them in varied contexts, and indeed this is borne out by the interview findings reported earlier (in 7.2.7). The notion that complex ideas can take time to bed in is therefore an extremely important one to communicate to children who are struggling to understand and to learn. It is also an important notion to communicate to some teachers, who are too eager to intervene the moment that any child appears to be floundering by providing the answer or substituting a simpler problem.

That most pupils' level of *competence* also increased when given opportunities for further practice is apparent from findings reported earlier in chapters 5 and 6, particularly from the coursework evidence on topics 9 and 10, therefore it can be concluded that pupils' feelings of confidence are well founded i.e. that their self-appraisals are accurate.

7.2.9 Pupils' appraisals of the resources

Pupils were unable to appraise the 'ideas to explore' within unit 3 and unit 4 topics. The class teacher advised pupils to omit them because time was running out. This has affected the intended balance of activities because there

ACCESS TO COMPUTERS (UNIT 1, QU. 11)

Were you able to get onto the computer whenever you needed to... ?

all of the time	3	15%
most of the time	17	85%
only sometimes	0	0%
	N = 20	

HELP HINTS (UNIT 1, QU. 8)
Only answer this question if you used the Help Hints.

Were the Help Hint useful?

very useful – I was able to fix most of my mistakes without having to ask for help	2	22%
quite useful – I was able to fix some of my mistakes without having to ask for help	5	56%
not useful – I still had to ask for help	2	22%
	N = 9	

ANSWER BOOKLET (UNIT 1, QU. 7)

Did you like using the Answer Booklet?

yes	15	83%
no	3	17%
	N = 18	

Give a reason for your answer.

REVISION QUESTIONS (UNIT 2, QU. 8)

Did the Revision Questions help you to understand the work?

yes	14	74%
no	5	26%
	N = 19	

Please explain your answer.

SUMMARY SHEETS (UNIT 2, QU. 9)

Were the Summary Sheets useful to you?

yes	14	88%
no	2	12%
	N = 16	

Please explain your answer.

IDEAS TO EXPLORE (UNIT 3, QU. 9)

Did you try out any of the Ideas to Explore?

yes	0	0%
no	10	100%
	N = 10	

If yes, indicate the Topic and Number and give your opinion of it.

Figure 29: Questionnaire responses (unit 1, qus. 7, 8, 11; unit 2, qus. 8, 9; unit 3, qu. 9 – appraisals of resources).

was less opportunity for pupils to exercise choice and to engage in open-ended exploration. For all other aspects indicated opposite (figure 29) pupils' appraisals were generally very positive.

7.2.9.1 Access to computers

All pupils indicated that they were able to get access to a computer whenever they needed to, 'all of the time', or, 'most of the time', and there is no significant difference in the responses of boys and girls (although two out of three of the, 'all of the time', responses were from boys, the numbers in this category are too small for comparison).

However, having to share access to computers was not regarded as satisfactory by everyone. Claudia found it annoying:

Cl: A few times when you have to wait and there's no written work you can get on with, you read instead. I think you have to read ahead [in order to understand the work], but it annoys you if you've written a program and you want to try it out.

[Interview 4, Claudia, 9/11/93, Transcripts p.29]

Responses to other questionnaire items also reveal some pupils' dissatisfactions, for example, Anthony had complained about the time taken to load programs because he sometimes had to log on to the system more than once during a lesson (unit 1, qu. 3), and one of Dawn's reasons for thinking it was better to work with a partner is that she and her partner would have guaranteed access to a computer (unit 1, qus. 3 and 9).

In the first phase of interviews, I asked pupils what they did when they were waiting for a computer to become free:

SC: Just read through what I had already done to make it a bit clearer in my mind.

MK: Did you feel that you were wasting time?

SC: No, really no, because it was just like you were revising for the next task.

[Interview 5, Scott, 29/11/93, Transcripts p.35]

Chris, Lorraine and Paula worked ahead instead of revising:

CH: I could always go and read the other topics and stuff like that.

L: ...there was writing that I could do.

P: You'd maybe have to go onto the next task to write out something...

[Interviews 1–3, Chris, Lorraine, Paula, Transcripts p.10, 17, 21]

Although the class teacher was pleased that pupils made productive use of their time, she was nevertheless very dissatisfied with the level of hardware provision. In her experience, pupils made more careless errors when entering, testing and debugging their programs because they felt under pressure to work quickly at the computers so that other pupils would get a chance to use them. Having to wait for a computer to become free disrupted pupils' concentration and was a disincentive to progress. She insisted, when a new lab was being set up in June 1995 (the timing was most unfortunate for this study!), that there should be sufficient computers for pupils to have individual access.

7.2.9.2 Help Hints

The help hints for unit 1 (which provided hints on debugging) were used by less than half the class. They were located beside computers so they ought to have been convenient to access. The pupils who didn't attempt to use the help hints may not have needed to use them, or they may have preferred to ask someone for help with debugging. Perhaps it was easier to ask for help, in comparison to having to locate the error in the help hints and then think about how to correct it (the hints didn't supply the answers). Also if pupils lacked confidence in their ability to solve the problem independently, then one would imagine that they would be more inclined to seek assistance. However a comparison with the responses to (2K) – were pupils confident or unsure about their ability to find and correct bugs in a program? – does not provide any evidence for a link between confidence and use of the help hints (of the four pupils who were not confident about debugging, two did use the help hints and two didn't).

Of the nine pupils who did use the help hints, Paula and Dawn found them to be most useful, Bryan, Claudia, Caroline, Anthony and Michael found them to be useful some of the time, and only Stephen and Chris indicated that they did not find them to be useful at all.

I asked Paula, Claudia, Chris and Scott about the help hints (in the first phase of interviews) and got a range of responses.

P: *...if the teacher was with someone else I used the help hints. They just told me what to do and I went on without actually having to go to the teacher every two minutes.*

CL: *I did [use the help hints] to begin with, 2 or 3 times, but later on I didn't need them so much since after a while I understood the work. I probably just forgot to use them. No one in the class used them at the end of the unit.*

CH: *I could have used them more but, every now and then I got stuck and I asked the teacher.*

SC: I didn't really find that I needed to use them ...

[Interviews 1, 3, 4, 5, Chris, Paula, Claudia, Scott , Transcripts p.9, 20, 29, 34]

Both Paula and Chris preferred to ask the teacher. Claudia's and Chris's responses suggest that pupils were not very used to using reference material of this sort to solve emergent problems.

7.2.9.3 Answer Booklet

The majority of respondents indicated that they liked the answer booklet. Their responses were categorized as follows:

A The answer booklet forms a neat, well organized record of your work (5 respondents);

B it is well designed and easy to use (4 respondents);

C it is good for referring back to (4 respondents);

D it helps you to keep track from day-to-day and to refresh your memory (4 respondents);

E it enables you to understand the work better (1 respondent);

F it cuts down on writing and saves you time (1 respondent).

The set of comments below was categorized under (A), (B) and (F), where an underlying theme is the *smooth and efficient progress of learning.*

Bryan: *The answer book is easy to use because it relates exactly to the task sheets.*

Paula: *It helped me to show answers neatly and clearly.*

Claudia: *It was easy to understand and quick to list [the] answers.*

Carol-Anne: *It was an easy way to display my answers neatly.*

Chris: *I could find things more easily.*

Caroline: *It enabled me to set out my answers neatly ...*

Catherine: *Because it was easy to write your answers and it was easy to use.*

Gerard: *It was easy to use and easy to understand.*

Lorraine: *It helped me to set out my answers neatly.*

The comments about neatness are all made by girls.

The next set of comments were categorized under (C), (D) and (E), where underlying themes are the *effective monitoring and review of learning.*

Claire: *It made sure that I understood the lesson.*

Dawn: *[I liked the answer booklet] because it reminded me [of] what I had done also if I was stuck I could look back to it.*

Caroline: *... and [it was] easy to understand for future reference.*

Michael: *Sometimes the teacher is busy and you need help.*

Bernadette: *I like using it because it reminded me [of] what I had just done on the computer.*

Kelly: *Because it helped you remember what you did [sic].*

Kathleen: *Because it was easier to remember what you had did [sic] the day before and you could write it down in your own way so that you understand.*

Kathleen's observation about writing things down in your own way indicates that she is probably contrasting this method of recording her work with copying down notes from the teacher or from a text book. She is stating a preference for active learning because it aids her understanding.

Only three pupils (all boys) indicated in their written responses that they didn't like the answer booklet:

Stephen: *Because it's irritating working from two different books.*

Anthony: *It didn't give enough details on some of the programs.*

David: *I didn't like the answer booklet because the questions were in another book so you couldn't really take it home.*

David could have taken his answer booklet home to do extra work because the class teacher had made separate copies of individual topics for pupils to take home (whether he could be relied upon to bring them back the next day is a different matter!).

Overall, the findings on the answer booklet point to its effectiveness as a learning resource for the majority of pupils in the class, and the findings also point more generally to the effectiveness of the learning environment in fostering independent learning.

7.2.9.4 Revision Questions

The majority of respondents indicated that the revision questions did help them to understand the work. Their responses were categorized as follows:

A the revision questions gave me the opportunity for more practice, and I could therefore feel more confident about my understanding (7 respondents);

B they helped with retention (3 respondents);

C they helped me to revise, or enabled me to put my knowledge to the test (1 respondent);

D no explanation given (e.g. 'they did help me to understand the work') (5 respondents).

From this it can be seen that some pupils interpreted the question openly, since there are references to revision and retention, as well as to understanding. Of the responses in categories (A)–(C), most (9 out of 11, or 82%) were from girls.

There is listed below a selection of the responses which ranges across the different categories.

Claire: *Some of it I don't understand in class but when I take it home and read over [it] a few times I start to understand.*

Paula: *If there was something I was not entirely sure about that it was right they helped.*

Kathleen: *It gave me more of an idea about things...*

Carol-Anne: *It was like studying for something, it gave me more practice for when I went on to the computer at class time.*

Anthony: *They broke up the work bit by bit.*

Kirsty: *It kept the things fresh in my memory [and] helped me go over the things I was having problems with.*

Caroline: *They 'go over' what you have already learned and 'test' your knowledge. It is a good way to revise and keep things fresh in your memory.*

In common with the other questionnaire items in which pupils are asked to make written appraisals, the responses provide insights into pupils' metacognitive awareness and their use of metacognitive strategies. An example is Claire's strategy of reading over the work a few times when she feels that she doesn't understand it properly, from which one can tell that she

is actively monitoring her understanding and taking appropriate actions to secure it.

Paula's and Kathleen's judgements about being sure or having an idea about something are nuanced: clearly their mental model of understanding is not an 'all or nothing' affair. In this respect, the responses are very similar to those of Carol-Anne, Kathleen, and Caroline when they re-appraised their confidence in the unit 2 questionnaire (see 7.2.8).

Carol-Anne refers to studying, not in the context of preparing for examinations, but in the context of coping better with her day-to-day learning in class. Caroline's and Kirsty's references to keeping things fresh in your memory are in a similar vein. Anthony's comment about the revision questions breaking up the work, 'bit by bit', indicates that their use helped to make a complex learning task manageable for him.

Claudia, Dawn, Bernadette, David and Gerard did not explain their 'yes' responses. Dawn has written, 'Yes it helped me to understand the work,' Bernadette has qualified her 'yes' response with, 'Sometimes,' and David has written, 'I was able to get a fuller understanding of the programs but...' (and he then moves on to a different theme). Claudia and Gerard failed to make any written comment at all. They may be *aware* that they have understood better as a consequence of doing the revision questions, but perhaps they cannot explain it very readily (at least, not without some prior discussion or further prompting).

David's reaction to the revision questions was mixed: 'I was able to get a fuller understanding of the programs but found that I didn't really have the time to do it [the revision questions] often. I would rather take my notebook home and do work which advanced me further.' This comment reveals David's anxiety about his progress, which is linked to his desire to get a credit grade (the same concern surfaces elsewhere in the data about David). His response brings to mind Gardner's (1993, p.24) warning about coverage being the enemy of understanding. Despite our efforts to curtail the extent of the syllabus by ignoring aspects of the content not strictly necessary for broad coverage, pupils do, unfortunately, still have to cover 'lots of things' to gain a credit grade for programming coursework. Nevertheless it is important for David to take his time in order to establish a proper understanding and to consolidate his learning, so that he can build a secure foundation for any subsequent learning.

David has to appreciate that trying to skip things will actually slow his progress later.

There were five 'no' responses (from Lorraine, Chris, Bryan, Scott and Catherine) which also fell into different categories, as follows:

E The revision questions didn't explain anything (1 respondent);

F ...but they *did* help with retention (1 respondent);

G no explanation given (e.g. 'They didn't make any difference to my understanding') (3 respondents).

Their written comments were:

> ***Lorraine:*** *It didn't really help me to understand the work but it did help me to remember the things I forgot.*
>
> ***Chris:*** *They didn't make any difference [to my understanding].*
>
> ***Bryan:*** *They didn't explain anything, they just asked similar questions to the work book. They didn't explain a thing.*
>
> ***Scott, Catherine:*** *[blank]*

When I asked Lorraine and Chris to expand on their answers[41], they said that they thought they already possessed a good level of understanding, which accords with their responses to Qu. 4 in the unit 2 questionnaire (Lorraine indicated that she understood the work 'all of the time', and Chris, 'most of the time'). Chris said that the main benefit for him of doing the revision questions was the extra reassurance that he gained from getting further practice.

Bryan seems to consider that the way for him to understand better is through being given more (or better) explanations. Perhaps he doesn't feel that 'doing questions' contributes much to his understanding (and it may not do if he arrives at his answers by serendipity, or if he copies someone else's work, or if he fails to review his learning adequately). His response does seem to be a plea for more individual assistance from the teacher.

7.2.9.5 Summary Sheets

The majority of respondents indicated that the summary sheets were useful to them. These are the categories which emerged:

---

41 Interview 6, Chris and Lorraine, 22/2/94, Transcripts p.51.

A The summary sheets were useful for *revision* and *consolidation* (7 respondents);

B they explained things well and helped me to *understand* the work better (5 respondents);

C they were useful to *refer to* if I was stuck or if I had forgotten something (3 respondents);

D they helped me to *remember* things that I had forgotten and refreshed my memory (2 respondents);

E they *condensed the information* well (2 respondents).

Of these responses, most (79%) were from girls.

There is listed below a selection of the responses which ranges across the different categories:

Scott: *They were helpful when doing the revision sheets.*

Caroline: *They act as 'back up' to the revision questions. They are good to refer to, if necessary.*

Carol-Anne: *They helped me to do the revision questions at home when I didn't have my folder to help me.*

Kelly: *I used them when I couldn't find the answers to the questions in the revision questions booklet.*

Michael: *I was learning it twice.*

Lorraine: *They helped me to understand it a bit better and also revised what I'd forgotten.*

Kathleen: *They explained how to answer the questions. They are good for looking over what I have done. They help to refresh my memory.*

Claudia: *If I ever got stuck I looked at them.*

Paula: *If there was something I had forgotten how to do from the beginning of the topic it helped [to use the summary sheets].*

Kirsty: *The summary sheets condensed the information and made it easy to understand.*

Bernadette and Gerard both indicated that the summary sheets were not useful to them, but only Bernadette gave an explanation, in which she said that she didn't understand them. Bernadette also found the task sheets for unit 2 quite difficult to follow, and she assessed that she understood the work in the unit

only, 'some of the time'. The summary sheets may not have helped her much because the explanations within them did not differ substantially from those in the task sheets. Three pupils (Bryan, Catherine and David) did not respond to the question.

The summary sheets were intended to assist pupils to do the revision questions at home, as well as to be available to pupils as a source of reference during class. However two pupils – Catherine[42] and David – did not use them, as I discovered during interviews. I had asked David his opinion of the summary sheets:

DA: What are the summary sheets?

MK: Those are the things that you are supposed to take home with the revision questions.

DA: Oh right, them! No I don't use them.

MK: I thought that was the case, because when I was looking at your homework questions [I noticed that] you tend to miss a lot out...What happened to your summary sheets?

DA: I don't know. I just thought they were for like, just there, just an extra bit of paper you had in your folder.

MK: Did it not occur to you that they might be useful for doing the revision questions?

DA: No.

MK: Given that you [now] know they are there, how will you use them in future?

DA: I think I'll just use them for my homework.

MK: What will you do? Will you read them before you do the revision questions?

DA: No, do the revision questions and if I was having trouble then I would have a look at them.

[Interview 11, Dawn and David, 20/6/94 and 21/6/94, Transcripts p.131]

Paula, Caroline and Kathleen did make use of the summary sheets when doing their homework, and they had indicated that they found them to be useful. I explored with them how and why they had used the summary sheets, and in what ways they had proved to be useful:

MK: Did you start by reading the summary sheets first, or did you do the revision questions first?

P: No, I read the summary sheets through for the topic then after I did that I went on to the questions.

---

42 Catherine said that she had not done any homework for unit 2 because her teacher had still to return her homework jotter to her.

MK: *What about you, Caroline?*

CI: No, I sort of just skimmed through them then did the questions so that I knew where things were when I needed to look back.

MK: *What did you do, Kathleen?*

KA: I read the summary sheets first then did the questions.

MK: *[Addressing Kathleen] Do you think that's a good strategy?*

KA: Yes.

MK: *Do you think you would have coped with the revision questions if you hadn't had the summary sheets?*

KA: Some things. Most of them, [but] I would have missed out some of them probably. [Others nodding agreement]

MK: *So there were certain things that you would have forgotten?... What kinds of things would you have been most likely to forget?*

P: The string variables and the numeric variables.

MK: *What about you, Caroline – what do you think?*

CI: Sometimes it would give you [in the revision questions] the beginning of a program or something and you had to fill in the rest. But then if you look back at the summary sheets they would maybe have a design or an example – like you could look back and get an idea of what you were doing.

MK: *Paula?*

P: It showed you the main points of what you were doing ....

[Interview 7, Paula, Kathleen and Caroline, 9/5/94, Transcripts p.62–63]

Scott, Bryan and Carol-Anne also made use of the summary sheets, and they too indicated that they found them to be useful. I asked Scott:

MK: *In what ways [were the summary sheets useful]?*

SC: If you weren't sure you could just look them up....

MK: *When you were doing the revision questions, did you actually look at the summary sheets?...You did, Scott – just if you were stuck? Carol-Anne?*

CA: Only if I was stuck.

MK: *Bryan?*

B: I think only if I was stuck.

MK: *You didn't actually read them through first before you started [the revision questions]?*

CA: No.

MK: *Was there a reason for not reading them through first? Carol-Anne?*

CA: I don't know, I just didn't.

MK: *Scott, did you think, 'I'm not going to read the summary sheets first', or did it not occur to you [to think about it]?*

SC: It didn't occur to me.

MK: *If I did suggest to you, 'You should read them [the summary sheets] first before you do the revision questions', would you think that was a good suggestion?*

CA/B: Probably.

MK: *Why?*

CA: I don't know.

B: Because you forget, when you've done a topic you just forget – you can't remember specific programs.

CA: Especially if you've done it a wee while ago and [you're] trying to remember.

MK: *Could you give me an argument for* not *reading the summary sheets first?*

CA: Just trying to prove you remember.

B: You test yourself.

[Interview 10, Carol-Anne, Bryan and Scott, 14/6/94, Transcripts p.107–108]

When one looks across these responses, it is apparent that David has given little thought to how he could accomplish the homework well, whereas Paula, Caroline and Kathleen have set about the task in a very purposeful and planned way – they are aware of the types of difficulties that they might encounter with the homework, such as forgetting the main points, and being uncertain of how to proceed. The investment of their time (albeit that Caroline has decided to skim the summary) to review the key points of the topic or to look at examples of solutions before attempting the questions is one that they perceive as worthwhile. Bryan, Scott and Carol-Anne have waited until they were stuck before referring to the summary sheets (and this is also the strategy that David plans to use in future), but they don't seem to have made any conscious decision, before they embarked on the homework, about how best to utilize the summary sheets in order to enhance their learning and performance – they were not *strategic* in their approach.

Boekaerts distinction between 'awareness', and 'willingness' to use personal resources in the service of a learning task, is relevant to this situation (see 2.6.2). There is no doubt that, in this case, one of the factors which would have

WORKING ON YOUR OWN OR WITH A PARTNER (UNIT 1, QU. 9)

Do you prefer to work ...?

on your own	13	72%
with a partner	5	28%
	N = 18	

Give a reason for your answer.

WORKING AT YOUR OWN PACE (UNIT 1, QU. 10)

Did you like being able to work at your own pace?

yes	20	100%
no	0	0%
	N = 20	

Figure 30: Questionnaire responses (unit 1, qu. 9 – own or with a partner?; unit 1, qu. 10 – working at own pace).

affected pupils' personal appraisals of how worthwhile it was to devote time and careful thought to doing the revision questions was that they had to be done in their own time at home. Another is that it would not contribute directly to the final grade. The key issue is whether appraisals can be influenced positively by the learning environment and the teacher's actions.

7.2.10 *Preferences for working on their own or with a partner, and for working at their own pace*

In chapter 4 (sub-section 4.5.2), I described how pupils worked individually at computers when entering and testing their programs, and the rest of the time they worked at their desks which were situated in the centre of the room. Pupils could decide whether to work closely with a partner (or partners – sometimes there were three pupils seated together), or whether to work on their own. At this point it is important to emphasize that pupils were not set group tasks to do, rather they were encouraged to collaborate on individual tasks.

The majority of pupils expressed a preference for working on their own rather than with a partner. However two pupils, Kelly and Catherine, modified the questionnaire to add a third category, 'Both' (which, for the purposes of collation, I have classified as an invalid response). I shall return later to discuss Kelly's and Catherine's reasons for preferring both ways of working. Everyone in the class liked being able to work at his or her own pace (figure 30).

Those pupils who preferred to work *on their own* gave the following reasons:

A I prefer to work on my own because you can work at your own pace (9 respondents);

B there are fewer interruptions or distractions (4 respondents);

C it is easier to self-monitor and you can understand/learn better (4 respondents);

D you can be more independent or self-reliant (2 respondents);

E you get more time on the computer (1 respondent).

Those pupils who preferred to work *with a partner* gave the following reasons:

Own	***Bryan:*** *...you can work at your own pace.*
	David: *I get the work done at my pace and not someone who could be slower.*
	Paula: *You can go at your own speed without having to rush your work to keep up with a partner or wait for your partner to finish.*
	Kathleen: *...I get a lot more done and anything that goes wrong is my fault. I don't talk as much if I work on my own.*
	Scott: *...you can work without being interrupted and ... at your own pace.*
	Anthony: *It is easier to get on with your work.*
	Kirsty: *I could get on with the work without being distracted or held back.*
	Caroline: *...I can work in my own way and design things the way I like and find interesting.*
	Lorraine: *If you're stuck and want to take your time you can. You can do your own thing.*
	Claudia: *You can work at your own pace, and it's your own ideas and mistakes so you know where you're going wrong.*
	Chris: *If I get it wrong, I can try again on my own.*
	Bernadette: *[There is] no-one to get you mixed up.*
	Carol-Anne: *...I can work at my own pace, learn better from my mistakes and I can get more time working on the computer.*
Partner	***Claire:*** *If you are stuck, then they could help you and it doesn't feel as though you're doing a lot of work.*
	Dawn: *...both people would be on the computer all the time, you could do the work twice as fast. If you needed help you could ask your partner instead of the teacher all the time.*
	Stephen: *It's easier [with a partner].*
	Michael: *You don't make as many mistakes.*
	Gerard: *You can make sure you are on the right track.*
Both	***Kelly:*** *...on my own I could work at my own pace and when I worked with a partner I could help her and she could help me.*
	Catherine: *...you can discuss what you're going to do with each other or if your partner is off then you can do it yourself.*

Figure 31: Questionnaire responses (unit 1, qu. 9 – own or with a partner?, reasons for preferences).

F I prefer to work with a partner because you can help each other if you are stuck (2 respondents);

G you can share the workload and this makes it seem easier (2 respondents);

H you can spot errors and it is easier to stay 'on track' (2 respondents);

I you get more time on the computer so you make faster progress (1 respondent).

All of the responses are shown in figure 31 opposite. This item was one of the most successful at probing pupils' self-knowledge. Most pupils appear to display a *diligent and responsible attitude* to their work (reinforcing earlier impressions, for example, when pupils worked on or revised while they waited for a computer to become free). Underlying the desire to have control over the pace of learning (the category with the highest frequency of response) is the concern in many cases to *understand* (e.g. 'If you're stuck and want to take your time...'), and in others to *make progress* (e.g. 'You can go at your own speed without having to...wait for your partner to finish.'). Being able to *concentrate* on the work is a recurrent theme; so also is *effective monitoring of learning* and the ability to *recover from errors*. Clearly, many pupils in the class view their progress and the quality of their learning as being significantly affected, either adversely or favourably, by their interactions with other pupils.

Some of the responses are refreshingly honest – how many adult learners will admit to getting mixed-up, making mistakes or being too talkative? Chris spoke of avoiding the temptation to copy another pupil's work when I asked him to explain his preference for working on his own:

> ***Chris:*** *Well, if I was wi' a partner he might do a task, right, and I would maybe, see, think I was doing it wrong, and I'd just copy him maybe so that I'd get it right if I wasn't sure. But if I was on my own I would need to think about it more, make sure I did it right and I'd know what to do.*
>
> *[Interview 1, Chris, 15/10/93, Transcripts p.10]*

Thinking about what he is doing and doing it properly seem to matter a lot to Chris; his response is very indicative of a deep rather than surface approach to learning. Paula has responded in a very similar way when she reflected on the benefits of doing the revision questions at home (see 7.2.8).

Kathleen's, Claudia's, Carol-Anne's and Chris's attitude to making mistakes is matter-of-fact; the important thing is to be able to unravel and fix them, and also, according to Carol-Anne, to learn from your mistakes. Papert's (1980) assertion that computer programming can lead children towards this philosophy by gradually undermining their resistance to debugging would seem to hold true for them. Michael's and Gerard's attitude to mistakes is also a pragmatic one; they are best avoided in the first place (and they consider that having a partner is helpful in this respect). Very few bugs were present in pupils' finished programs, indicating that they did indeed succeed in unravelling and fixing their errors.

There was a higher proportion of boys among the group who indicated a preference for working with a partner. The three boys who fell into this category seemed, from their comments, to be uncertain about their ability to manage the work so well on their own. Dawn and Claire, who also indicated a preference for working with a partner, were among the least confident about the work by the end of unit 2. In contrast, all of the pupils who got the top grade for programming coursework (Paula, Chris, Lorraine, Caroline and Kirsty), and who were, on the whole, confident about their abilities, had expressed a preference for working on their own. Caroline's response signals her confidence and self reliance, '...I can work in my own way and design things the way I like and find interesting.' Pupils' self-assessments of their programming abilities may have influenced their choice of whether to work with a partner or on their own.

It was my impression from working with the class, from examination of their work, and from interviews, that there was far more collaboration actually going on between pupils such as Paula and Lorraine, Bryan and Scott, and Kathleen and Claudia, than their responses in the unit 1 questionnaire seem to indicate. For example, although Paula had indicated a preference for working on her own, I asked her if she and Lorraine, who sat beside her all through S3, worked together on problems (it turned out, to my surprise and also the class teacher's, that they were cousins!):

P: We did – I think we did that at the beginning, then we worked on our own. It was okay – I like Lorraine, she's my cousin – so it was good working with her, but I prefer to work on my own because you can go at your own speed. Sometimes she goes faster than me, then she has to wait on me – or if I go faster than her on something, then she's still trying to write something out...

MK: So did you both decide that you would just work ahead at your own pace?

P: Yes.

MK: *Were you still able to help each other...?*

P: Yes. If there was something wrong and we didn't understand anything or did something wrong – we asked each other – spoke to each other. But one time she put in a lot of her program in as comments after the flashes[43]. She was wondering why it wasn't printing! [laughs] Then things I did – if I made a mistake, she'd point it out – 'You've done that' – and I'd go back and fix it. Things that you don't really notice on your own.

MK: *Did you just tell her what was wrong, or did you help her to find out for herself?*

P: I just told her!

MK: *Did she appreciate you doing that?*

P: Yes.

MK: *Do you think that, although you just told her what was wrong, she* understood *what was wrong?*

P: I think so [sounding hesitant].

MK: *I take it, it worked the other way round sometimes – that she would be helping you? Did that happen too?*

P: Yes.

[Interview 3, Paula, 1/11/93, Transcripts p.20–21]

It seems from this account – and also Lorraine's[44] – that they had found a productive way of working together to give them some measure of independence from each other (and also a measure of independence from the teacher, since they were able to help each other). The benefits were not just that they were able to help each other out when they were stuck, but they were also able to monitor each other's performance (with the added bonus of having an occasional laugh at the other's mistakes!).

It could be that pupils in the class collaborated more and better as their relationships formed and strengthened over time and they began to establish a pattern of working together (this process would come about more quickly for Paula and Lorraine since they already knew and liked each other). The positioning of this item in the unit 1 questionnaire and not in the later questionnaires may have failed to capture this.

---

43 The 'flashes' are two parallel lines ('//'), which signify that a comment has been inserted in the program. The COMAL interpreter will ignore any comments. Hence if they are placed accidentally at the start of each line of the program, nothing will happen when the program is run.

44 Interview 2, Lorraine, 15/10/93, Transcripts p.16.

PROGRESS

UNIT 2 (QU. 10)

You have now finished the programming work at Foundation level and have started on General level work (there are three more topics at General level).

Are you pleased with your progress?

yes	10	53%
no	5	26%
not sure	4	21%
	N = 19	

UNIT 3 (QU. 8)

You have now finished the programming work at General level and have started on Credit level work (there are four more topics at Credit level).

Are you pleased with your progress?

yes	8	80%
no	0	0%
not sure	2	20%
	N = 10	

Figure 32: Questionnaire responses (unit 2, qu. 10; unit 3, qu. 8 – progress).

Kelly's response seems to indicate that she sometimes worked on her own and sometimes worked with a partner. Catherine's response seems to point more towards working with a partner (it is only if her partner was absent that she would work on her own). All of those pupils who indicated a preference for working with a partner, and also Kelly and Catherine, had also indicated (in response to Qu. 10) that they liked being able to work at their own pace, and presumably they meant by this that the pace was not dictated by the teacher, as would occur with lock-step teaching methods.

Overall, the evidence suggests that most pupils were able to operate flexibly in their interactions with other pupils, maintaining a pace of learning that suited them individually, but nevertheless being able to discuss the work, compare answers, combine resources, or get help with problems. It could be that the nature of the task itself would determine in part how productively pupils could share in it and collaborate, however I did not attempt to probe to this depth.

7.2.11 Satisfaction with progress

Pupils were asked to indicate if they were pleased, displeased or uncertain about their progress as they reached the end of units 2 and 3. This question was not asked in the unit 1 questionnaire because it was felt that most pupils would not have given much thought to their progress at such an early stage of the course. The results are shown opposite in figure 32. Just over half of respondents indicated that they were pleased with their progress by the end of unit 2, in comparison to 80% by the end of unit 3.

The 'no' or 'not sure' responses were as follows:

Unit 2: Bryan, Michael, Bernadette, David, Gerard ('no'); Claire, Scott, Dawn, Anthony ('not sure')

Unit 3: Paula, Caroline ('not sure').

Of those pupils who were not pleased or not sure about their progress by the end of unit 2, only David managed to complete the next unit (and, having done so, he was then pleased with his progress), and Claire was half-way through topic 14 when, unfortunately, time ran out for her. Since the first credit level assessment is at the end of unit 3, none of this group gained a credit award apart from David. Of those pupils who did gain a credit award, 9 out of 10 (90%) – i.e. everyone except David – were pleased with their progress by the

end of unit 2; the corresponding statistic for those gaining a general award was 1 out of 9 (11%)[45], which is quite a contrast – Carol-Anne was the only one of this group who was pleased with her progress at the end of unit 2.

The reasons for pupils' concerns about progress were discussed during interviews. Claire and Bernadette were very concerned about their progress by the end of unit 2:

MK: *Bernadette, you're not [pleased with your progress], why is that?*

BE: Because I don't feel I know what I'm doing. Do you know what I mean? If I had time to read it all through slowly – and take it in.

MK: *Claire, you are not sure about your progress – why do you say that?*

C: I don't know. I'm not happy with it.

MK: *Do you think you are not doing well enough?*

C: Yes, like compared to a lot of the people in the class – everyone is away on topic ..., I don't know – I'm just a bit behind, I think.

BE: You're more [further on] than me!

[Interview 9, Catherine, Bernadette and Claire, 24/5/94, Transcripts p.89–90]

Bernadette is concerned about her lack of understanding. She has tried to keep up with others in the class, and as a result her understanding has suffered. Claire compares her progress unfavourably with that of other pupils. They both did remarkably well to reach as far as they did in unit 3, which is a testament to their diligence and determination. Bernadette's comment to Claire (above) is also indicative of the support and friendship which they extended to each other.

Stephen was the furthest behind in the class, and he was unhappy about this and the time that it was taking him to work through some of the problems. There were many factors that could be identified, from his discussion with me, as contributing to his slow progress relative to other pupils in the class.

- boredom

  MK: *But you found the earlier work [in Unit 1] very boring?*

  S: Yes.

  MK: *What made it boring for you?*

---

45 This result is statistically significant at 0.1%, using a two-tailed Chi-squared comparison.

S: I don't know, it was just all the same – just sitting printing programs out all the time.

- confusion

S: Sometimes I just couldn't follow the .... [unclear], you know, how you have to jump back sometimes and everything.

MK: Give me an example of what you mean.

S: When I done [sic] that, [looks up page 47 and points to Task 1 of Topic 10, which is writing out the design for the address cards problem] I had to keep going back to look at all this stuff.

MK: So when you were doing the design for the address cards you were referring back to the design for ....

S: Posters.

MK: You found that made it harder to do the work?

S: Yes, to keep going back all the time. I kept getting mixed up at what bit I was looking at.

MK: You were getting confused, then, between the two problems?

S: Yes, ended up for address cards I kept writing posters in it, and everything.

- forgetting filenames and losing programs

S: ...I saved it hundreds of different times, then I was loading in one, then mistakes in others – in the end I couldn't find my good one.

MK: Right so you had saved different versions of the program, some of which didn't work?

S: Yes.

MK: Then you would have to spend a lot of time looking through for the correct version of the program?

S: Yes.

MK: I'm just trying to think if there is any way that could have been avoided. Why do you think you were building up all these wrong versions of the program – were you saving them all using different filenames?

S: I think so.

MK: It usually tells you, save your program as something or other, and normally if you did that, then each time you saved it you would overwrite the previous version. You must have been saving your programs using..

S: Numbers and everything, and letters.

MK: .... a variety of names. Did you have version numbers for them, is that what you did?

S: Yes, but then I got them all mixed up. [46]

MK: Why do you think that you wouldn't remember from one day to the next which program you had been working on previously?

S: They are all dead similar, the names I'd saved them.

- not keeping any ongoing record of progress

MK: Did you keep the progress record at the start of your answer booklet?

S: No, I've not kept that for ages.

MK: Why did you decide not to do that?

S: I didn't have time at the end of the period some days, we never take a couple of minutes to stop and fill it in.

MK: Were you reminded to do that?

S: I think she [the teacher] just expected us to do it.

- frequent absences

MK: Do you think it's because the work is difficult [that you are experiencing problems], or do you think it's more to do with losing track of what you are doing?

S: I think I just lose track. See if I'm off for one period I find it dead hard to catch up and get back to where I was.

- not understanding the work

S: Some of it I just don't know where I am.

MK: What do you do when you don't know where you are?

S: Look through that [points to Unit 2 Task Sheets]... If I ask Mrs xxxx she tells me to look through it, because it will definitely tell me, if she knows that it will.

MK: Why do you think she is doing that?

S: So that I learn it and remember it.

MK: Do you find that you remember it better if you have been able to sort it out for yourself?

S: Most of the time.

MK: Okay, but it's taking you longer then because you have had to go back and read things again.

S: Yes.

---

46 An example of this is 'POSTER', 'poster1', 'POSTERS1', 'POSTERSIII' – from a printout of his user area on the network hard disc.

- finding it difficult to get to grips with technical vocabulary

MK: *You found the instructions quite difficult to follow as well?*

S: Yes.

MK: *Do you think the explanations are difficult to understand, could they be explained better?*

S: I don't know.

MK: *Is the language difficult for you to read?*

S: No.

MK: *It's not that. So you think the language is quite clear?*

S: Yes the language is clear.

MK: *Is it the technical words within it like top down design, top level refine ....*

S: Yes. I think that's what gets me most mixed up since I started in computing – all the names for different things, that's when I get mixed up.

- missing out the revision questions

MK: *How come you didn't do them?*

S: I don't think I took them out my folder [which is kept in class].

MK: *Do you think it might have helped you to do them?*

S: I don't know, sometimes if I take it home I just get more confused. [Presumably he is referring here to the Task Sheets i.e. the work he was doing in class]

MK: *Do you think you were having so much difficulty with the work that you wouldn't have been able to do the homework?*

S: I got wee bits of it but I would just have got dead frustrated and stopped.

MK: *Were you worried about that – was that your reason for not doing it?*

S: I think so, I think I just knew that there wouldn't be much I could do anyway.

- needing more opportunities to discuss the work with someone who can help

MK: *This is the first time that I've spoken to you [in an interview!], I've spoken to some of the others two or three times during their work. I think it's because I planned to speak to people at certain stages as they completed units and you've now just got to the end of Unit 2. Might it have helped if you had more opportunities to speak with me as you worked through?*

S: Yes, I think so.

MK: *In what way?*

S: It makes it clearer. It's easier to grasp something if you are talking about it.

- ...and messing about in class!

MK: So there has not been enough time to sit down and talk to me or Mrs xxxx to properly explore the things that you are not sure about?

S: And with everybody else, messing about in the class as well.

[Interview 20 with Stephen, 28/2/95, pages 246–270]

Stephen's answers give the impression that he would have preferred to work on straightforward but varied problems that could be easily accomplished quickly, so that there was less need for him to monitor progress from day-to-day or to review earlier work. He might also have preferred whole class teaching rather than to have to work from the materials; at least he would not have fallen behind to the same extent (although there may have been gaps in his knowledge due to absences). Whether this would in fact have helped him to cope better with the work and to settle down in class is difficult to tell.

Gerard and Michael were also disappointed by their progress, and, like Stephen, they were amongst those pupils who had made the slowest progress. Gerard cited his behaviour ...

MK: Very briefly, what disappoints you about your progress?

G: It's not the work or anything, it's my behaviour I think.

MK: Do you think you could have focused on it [the work] better?

G: A lot better, yes.

MK: Why was that?

G: I get distracted.

[Interview 13, Gerard, 2/12/94, Transcripts p.162]

and Michael frequent absences, as the reason for falling behind.

MK: You are not pleased with your progress. Can you explain...?

M: Because I was off a lot and...

MK: Do you think you got behind?

M: Yes.

[Interview 19, Michael, 14/2/95, Transcripts p.241]

LOOKING FORWARD TO NEXT UNIT

UNIT 1 (QU. 13)

Are you looking forward to doing Unit 2?

yes	16	80%
no	1	5%
not sure	3	15%
	N = 20	

UNIT 2 (QU. 11)

Are you looking forward to doing Unit 3?

yes	13	68%
no	3	16%
not sure	3	16%
	N = 19	

UNIT 3 (QU. 10)

Are you looking forward to doing Unit 4?

yes	6	60%
no	1	10%
not sure	3	30%
	N = 10	

Figure 33: Questionnaire responses (unit 1, qu. 13; unit 2, qu. 11; unit 3, qu. 10 – looking forward to next unit?).

7.2.12 *Were pupils looking forward to doing the next unit?*

Pupils were asked to indicate if they were looking forward to doing the next unit, if they were not looking forward to it, or if they were not sure. The results are shown opposite in figure 33. The proportion of respondents giving positive responses decreases from unit 1 to unit 3, from 80% to 60%. The 'no' or 'not sure' responses were as follows:

Unit 1 questionnaire: Stephen ('no'); Claire, Catherine and Gerard ('not sure')

Unit 2 questionnaire: Bernadette, David, Gerard ('no'); Claire, Scott, Dawn ('not sure')

Unit 3 questionnaire: David ('no'); Chris, Caroline, Lorraine ('not sure').

There is a close correlation between pupils' responses to this question, and their responses to the question on progress (see 7.2.11). Bernadette, David, Gerard, Claire, Scott and Dawn each responded in an identical way to the question on progress in the unit 2 questionnaire. Clearly their attitudes towards the course were closely bound up with their feelings about their progress. Some pupils did not *feel* that they were succeeding because they were falling behind the others in the class. However this wasn't the case for everyone; Bryan, Michael, Anthony and Paula had each indicated that they were either unhappy about their progress or uncertain about it, but nevertheless they were still looking forward to doing the next unit. Michael explained that he liked some of the unit 3 problems that other kids were working on, such as the nursery tunes program that Chris was doing[47], and Bryan seemed to be in agreement with Scott that the 'half decent' problems were all in the later units[48], and therefore they were looking forward to doing the next unit. Although there is no data on Stephen's views on his progress by the end of unit 1, his response to question 13 can be predicted from his other responses to the unit 1 questionnaire.

The responses also correlate well with those for item 4 in the unit 2 questionnaire, in which pupils assessed their understanding (see 7.2.7). All of those pupils who assessed that they understood only, 'some of the time'

---

47 Interview 19, Michael, 14/2/95, Transcripts p.242.

48 Interview 16, Scott and Bryan, 20/1/95, Transcripts p.199.

(Bernadette, Gerard, Claire, Scott and Dawn) were also not looking forward to, or were uncertain about, progressing to unit 3. Bernadette and Claire told me that they were fearful that the work would get harder and that they would no longer be able to cope with it[49]. Given Dawn's lack of confidence about the techniques that she had learned in unit 2, it is quite likely that she would also be apprehensive about the next unit (she was unfortunately absent on the day that I had arranged to continue with her unit 2 interview and so I did not get the opportunity to ask her directly).

Caroline also expressed some anxiety about the difficulty of the work in unit 4. She said, 'It just got a lot harder at the end [of unit 3] ... and I thought if it's hard now it will be even worse then.'[50] On the other hand, Catherine seems to have found her feet as the course has progressed, and she is no longer in the 'not sure' category by the end of unit 2. She explained that she anticipated that the work would become more interesting in unit 3[51].

Some pupils had actually attained their targets when they finished the unit they were working on, or there was little time left to continue beyond it. This was the case for Claudia, Kelly, Catherine, Kathleen and David when they finished unit 3. Nevertheless they had each, apart from David, indicated that they were looking forward to continuing.

A point which showed up in the third phase of interviews is that a few pupils – Scott, Bryan and Caroline – complained that the programming work had gone on for too long.

7.3 SUMMARY AND DISCUSSION

The summary and discussion of the questionnaire and related interview findings will focus on the following aspects: (1) the quality of the data obtained; (2) the overall reaction of the class to the learning environment; (3) differences and trends – contrasting reactions; gender differences; in the reactions of pupils with high attainment (credit grades) and average attainment (general grades); (4) constraints affecting the quality of pupils' learning and the research outcomes.

---

49 Interview 9, Catherine, Bernadette and Claire, 24/5/94, Transcripts p.91.

50 Interview 25, Caroline, 20/3/95, Transcripts p.357.

51 Interview 9, Catherine, Bernadette and Claire, 24/5/94, Transcripts p.91.

7.3.1 *The quality of the data obtained*

The *questionnaire data* has the following characteristics: *internal consistency,* which suggests that the case study pupils have responded seriously to the questions and have answered thoughtfully; a high level of *consistency with the other data* on progress and attainment, which indicates that the case study pupils were able to gauge their progress and attainment accurately; and *completeness,* since there are very few missing responses. When supplemented with other (related) findings from interviews, a clear profile emerges of the class and of how individuals within it reacted to the learning environment. The questionnaires have proven to be a very effective research intervention, and also a very effective pedagogical tool to prompt pupils to review their learning.

Pupils' responded very openly to questions. Some criticisms were made of aspects of the learning environment, but they were often accompanied by constructive suggestions for improvements. Only one pupil (Stephen) was very negative. Resnick (1987) has highlighted the importance of the social setting to foster the disposition to engage in higher order thinking. The social setting seems to have been effective in promoting independent thinking. In particular, the class teacher was a very active promoter of open dialogue ('your opinions matter,' she would say to pupils), especially among the more reluctant learners in the class, to try to involve them more.

The questionnaires provided a very satisfactory basis for further discussion during individual or group interviews. No difficulties arose over issues of confidentiality (or none that I am aware of), and, although pupils were asked to write their names onto the questionnaires, this does not seem to have prevented them from expressing their views honestly.

The *interviews* were intended to provide a context for pupils to review their learning and to strengthen their understanding. The serious manner in which pupils participated in them and the quality of their contributions resulted in both of these outcomes being met. Pupils listened and contributed well throughout (although I did notice that Scott was very quiet in the group setting and gave much better responses when interviewed on his own). Pupils settled quickly into the interviews and they soon seemed to forget about the tape recorder. The general impression that I had was that most pupils welcomed the interviews as a break from the normal classroom routine, and they responded well to being asked quite searching questions.

During group interviews pupils seemed to interact very naturally, and I am certain that this resulted in better evidence. Also my role as a mediator, to draw out the different learning and problem solving strategies that pupils adopted so that they could each gain better insights into their own processes of learning and problem solving, appears to have worked successfully.

7.3.2 The general reaction of the class to the learning environment

Although there were some contrasting reactions, the general response to the learning environment has been very positive.

The aspects of programming which pupils most liked were the opportunities for thinking independently and using your imagination, learning and practising new techniques, experiencing a sense of achievement and satisfaction, and producing solutions to 'real' problems. It is inconceivable that pupils would have responded in these ways had they been exposed instead to the dull and unimaginative coding exercises that characterize introductory programming courses in many schools.

The disliked aspects of the course for some pupils were laborious tasks (too much writing, for example, or keying in long programs), repetition, or work that was experienced as too difficult. Programming is labour intensive and requires close attention to detail. The programming language (Acornsoft COMAL), in comparison to more recently developed languages, has rather limited editing, design and debugging tools (however Scottish schools are gradually moving towards better programming environments as they upgrade their computing facilities). It is evident from the findings that some pupils generally need more reinforcement than others to grasp programming ideas and to become competent, and that individual pupils need reinforcement on different aspects (for example, the only real difficulty that Chris encountered was with design). Getting the right balance of incremental and practice tasks for each individual is not easy. The teacher has to gauge in an on-going manner the most appropriate 'next steps' for each pupil to take.

The general indicators on interest, understanding and confidence are very positive; indeed some pupils displayed a remarkably high (and justifiable) level of confidence in their programming abilities. One might have expected initial interest to wane since the subject matter was new and therefore novel to the majority of pupils, but this did not happen, except in one case (Michael). Not all concepts and techniques were easily understood when first

encountered, e.g. readable and user friendly programs, variables and loops. Also not all techniques were easily applied when first encountered, e.g. top-down design (see, for example, the discussion in 6.4.3). Pupils overcame difficulties with understanding mainly through getting assistance (either from the teacher or another pupil), re-reading the materials and retracing their steps, trial and error, or 'keeping going' – getting more practice on similar problems to strengthen their understanding. Pupils gained in confidence as they had further opportunities to apply particular techniques in a variety of contexts. When assessing their confidence, many factors emerged as important considerations: understanding; mastery of skills; retention of skills; durability of skills (or long-term retention); transfer of skills; the ability to carry out a performance unaided; the ability to recognize (without hints or cueing) how best to proceed when a new and unfamiliar problem is encountered. It is re-assuring that pupils applied such wide and rigorous criteria when assessing their competence and understanding, although it has to be borne in mind that this list represents the combined wisdom of pupils in the class, rather than the insights of any one individual. What this does point to is the enormous potential benefits of pupils sharing their insights and experiences as learners with each other.

Pupils' appraisals of the learning materials and other resources were generally very positive. All pupils could get *access to computers* most of the time when they needed to. If a computer wasn't free, pupils worked ahead (reading on, doing written work) or revised. Although the *help hints* were used by only nine pupils, they were regarded as useful by seven of these nine pupils (it would therefore seem to be worthwhile for teachers to promote their use). The majority of pupils liked the *answer booklet* because it enabled them to manage their learning more effectively. It was considered to be well designed and easy to use. The *revision questions* helped with understanding by providing opportunities for further practice, enabling pupils to put their knowledge to the test, and aiding retention. The *summary sheets* were useful: as an aid to revision and consolidation; because they condensed the information well, provided good explanations and helped with understanding; because they were useful to refer to when stuck; and because they reminded pupils of things that they had forgotten.

Every pupil liked to be able to work at his/her own pace. Although more pupils preferred to work on their own rather than with a partner, there was nevertheless much productive collaboration, according to observations and

interview responses (there are very few instances of copying in the evidence presented in chapters 5 and 6). It is also very evident (from interview responses) that the class teacher was someone to whom pupils could readily turn for assistance if it was genuinely needed. Pupils gave the following reasons for preferring to work on their own: being able to work at your own pace; avoiding interruption or distraction; being able to monitor (progress, performance, understanding) more effectively and to understand and learn better; and being independent and self-reliant. Pupils gave the following reasons for preferring to work with a partner: giving and getting help; sharing the workload; and spotting each others' errors in order to stay 'on track'.

One of the less positive indicators was on progress by the end of unit 2, where quite a high proportion of respondents (47%) were uncertain about, or displeased with, their progress. However most pupils were keen to learn more about programming. There were thirty-five 'yes' responses when asked, 'Are you looking forward to doing unit 2 [3, or 4]?', and only five 'no' responses; if pupils were unsure it was usually because they were worried that the next unit might be harder than the one they had just completed. A few pupils thought that the programming work had gone on for too long.

The cumulative evidence indicates that the majority of pupils were adopting a deep rather than a surface approach to learning – their concern was to understand and to be able to apply their knowledge successfully. However, as noted in the previous paragraph, rate of progress with the syllabus was also a very significant area of concern, and this indicates a strong achievement orientation. Many pupils' willingness to do programming in their own time (at lunchtimes, at home, and after school) indicates a determination to meet the targets that they had agreed with the class teacher.

There are many references in the questionnaire and interview findings to learning strategies (such as self-testing, monitoring, revising). One would expect to find, as evidence of metacognition, *purposeful deployment* of strategies, rather than their reproduction on cue, and *insightful* reflections. There is indeed evidence of purposeful and effective use of learning strategies, for example, when Paula, Caroline and Kathleen discussed how they tackled their homework (see 7.2.7). There is also ample evidence of pupils' ability to reflect insightfully on their own processes of learning and problem solving.

7.3.3 *Differences and trends*

7.3.3.1 *Over time*

As pupils have progressed through the units they have found the work to be more interesting, but also more difficult – levels of confidence about new aspects in later units have dropped off slightly for pupils who managed to complete all three units, and rather more steeply for the others. Since the intention when designing the learning materials was to present a sequence of problems of gradually increasing complexity, then this outcome is not unexpected (it would only be of concern if a substantial number of pupils were experiencing real difficulties with the later work and were losing understanding, which does not appear to be the case).

7.3.3.2 *Gender*

A number of gender differences emerged in the questionnaire findings (we also know from chapters 5 and 6 that girls made better progress and attained higher grades on average). To summarize, the main differences were: girls provided fuller responses to written questions when asked to review their learning and so we know more about their thinking and reactions; girls rated their interest, understanding and competence more highly; girls were more likely to respond with 'no' or a positive comment when asked if there was anything they didn't like about the work; girls liked problems because they got to learn and practise new techniques more often than boys liked problems for that reason; boys liked problems which resulted in useful and realistic programs more often than girls liked problems for that reason; girls complained about laborious tasks more often than boys did; boys complained about repetitive tasks more often than girls did; girls were more likely to perceive the benefits of using the revision questions and summary sheets; and girls were more likely to prefer to work on their own (rather than with a partner).

It is possible to 'explain' some of these differences by suggesting that the girls (as a group) were more focused on learning as an end-product, whereas the boys were more focused on achieving a working program – especially if they perceived the program as realistic or useful, with learning as a by-product. The girls considered that time spent on consolidation and revision was worthwhile, and they resented time spent on laborious tasks which prevented them from learning new techniques, whereas the boys tended to regard some

activities which were designed for consolidation as repetitious, and once they had achieved their goal (a working program) they were keen to move on to the next challenge. However to succeed in programming it is necessary to make effective recourse to previous solutions in order to arrive at new solutions efficiently. The girls have appreciated this (i.e. they have accurately assessed the task demands, which is a metacognitive skill) through purposefully setting out to establish a well integrated knowledge base that will be accessible to them in future. There may, of course, be other factors which have contributed to girls' better performance, such as social factors and maturity. From informal observations of the classroom, girls did seem to manage their learning well and collaborate effectively. The national statistics on performances across a range of Standard Grade courses indicate that girls generally outperform boys in practical problem solving and investigation (Ganson and De Luca, 1996), both of which call for pupils to work more independently from the teacher.

7.3.3.3 Attainment

When the questionnaire data is analysed separately by pupil attainment, comparisons in the following three areas were highly significant (the indicators were higher for pupils gaining credit grades than for those gaining general grades):

- pupils' assessments of understanding (unit 2);
- pupils' assessments of confidence (units 1 and 2); and
- pupils' satisfaction with progress (at the end of unit 2).

Some other (though less significant) differences emerged: on interest, and ability to follow the instructions in the task sheets (in both cases, the indicators were higher for pupils gaining credit grades), and preferred way of working (pupils gaining a credit grade were more likely to prefer to work on their own).

Attainment rather than gender appears to be the more significant factor in the findings, according to statistical comparisons on closed response items. Qualitative comparisons signify that metacognition is a significant factor in attainment. One example of this emerged when contrasting how different pupils approached the homework tasks (based on the revision questions), where the lack of a strategic approach affected the quality of some pupils' work. Another is Stephen's interview, where it is very evident that his inability or unwillingness to manage his learning on a day-to-day basis, amongst other

factors, affected his progress and performance adversely. The evidence on progress with coursework in topics 9 and 10 also indicates the importance of metacognition; many of the problems that pupils encountered having resulted from their failure to self-monitor, check and self-test (see 5.4). The assessment portfolio evidence reveals that some pupils failed to check and to revise their work, which resulted in them losing marks (see 6.4).

7.3.4 Constraints affecting the quality of pupils' learning

According to questionnaire responses there were various factors which slowed down progress, including pupils having to share access to computers, and the programming environment (Acornsoft COMAL) which made entering and debugging long programs a laborious process. Also the timing of public holidays and teacher inservice days (all of which fell on Monday's, Tuesday's and Friday's) upset pupils' work routines, made the scheduling of interviews more difficult, and reduced the available time.

There seemed at times to be two competing agendas, the first relating to pupils' acquisition of domain knowledge (programming concepts, principles, techniques, language features, process skills) and the second relating to the broader agenda of developing pupils' learning, problem solving and thinking skills more generally. If pupils were very concerned about their progress with programming or their understanding then this agenda tended to dominate. For some pupils, the main focus of interviews was on recollection and understanding of programming knowledge (i.e. much of the discussion focused around item 4 – overall assessment of understanding, and item 6 – confidence about a range of aspects).

It is natural that pupils will need more assistance when learning programming than when learning to use simpler tools, such as a word processor. Also with a problem-based methodology the situation arises naturally when pupils reach an occasional impasse with a solution, and they need assistance to think the problem through. Therefore the class teacher's and my judgements about when and how to intervene are crucial to the success of the learning environment. The majority of pupils will not learn to program well if left to their own devices to work through the resources.

7.4 CONCLUSIONS

The main conclusions arising from the analysis of questionnaire and related interview responses are indicated below.

1. Autonomy and responsibility for learning were promoted through the design of the learning and research environment. Most pupils were enabled to manage their learning effectively, and benefited from the added flexibility of classroom transactions in comparison to traditional 'teacher led' approaches. They operated flexibly in their interactions with other pupils, maintaining a pace of learning that suited them individually whilst still being able to discuss problems, compare solutions, combine resources, or get help with problems. However some were reluctant or ill-equipped initially to take on responsibility for their learning and, perhaps as a consequence, their affective responses were less positive than those of other pupils. These pupils also faired less well in terms of progress and attainment.

2. The questionnaires encompassed a wide range of indicators on pupils' affective responses to learning to program, and they were very effective as a tool to enable pupils to review their learning in an on-going manner. Pupils' self-assessments were formed by considering and applying a wide range of criteria that they themselves had identified, and their assessments were mostly accurate (according to other measures). The use of questionnaires, particularly when followed up by discussions with pupils, can provide an effective tool to gauge and further develop metacognitive skills.

3. The overall response to the learning environment and to the subject matter has been very positive. Extended written and verbal responses signify that most pupils have adopted a deep approach to learning, in spite of the pressures that many experienced to 'cover ground'. A pruning of the course content is called for in future years.

4. Pupils became more confident about particular techniques when they were given ample opportunities to practice them in varied contexts. The notion that complex ideas can take time to bed in is therefore an extremely important one to communicate to learners who are struggling to understand. It is an important notion to communicate to some teachers who are too eager to intervene by providing the answer or substituting a simpler problem. It is also an important notion to communicate to designers of programming (and other) courses who fail to appreciate the complexity of the ideas contained in their curricula and who attempt to pack too much content in.

5. Pupils' on-going assessments of their understanding, confidence and satisfaction with progress were the most significant affective indicators of summative attainment. Together they can provide a diagnostic tool to direct the teacher and pupil towards appropriate actions to strengthen learning and understanding and to enhance performance.

6. The girls as a group seemed to place more emphasis on learning rather than on task completion, in contrast to the boys, and this may explain why they have generally faired better with learning complex subject matter.

There are some aspects of pupils' learning which have still to unfold.

7. Pupils do refer to the importance of being able to apply knowledge and skills, but only occasionally to issues surrounding transfer of learning. How generalizable do they consider their knowledge and skills to be?

8. Which factors account for the differences amongst pupils in their affective responses, progress and attainment? Part of the picture is in place (there is strong evidence that metacognitive skills play an important role). Pupils' self-appraisals at the time of learning seem to have had an impact on performance. Their conceptions of ability (whether incremental or fixed/entity) may also have played a role. Is there any evidence that pupils' self-appraisals or ability conceptions have been influenced positively by this intervention?

9. For pupils whose progress is impeded by poor or erratic performance, how is the problem of tedium and frustration, which leads to further errors being introduced and further slowing of progress, overcome? The answer may not lie in easier work. Is there evidence in this study of successful approaches being used?

10. Pupils' responses to questionnaire items have naturally focused on their reactions to learning to program, but the responses have also been quite revealing about their approach to learning. Do pupils consider that they have gained from this experience any useful insights about learning as well as about programming?

8 CASE STUDY FINDINGS IV: REFLECTIONS ON LEARNING AND PROBLEM SOLVING

8.1 CHAPTER OVERVIEW

This chapter examines pupils' insights on learning and problem solving as revealed through their responses to review tasks and during interviews. It also presents some evidence on near transfer as revealed through practical problem solving performances using a spreadsheet. A central theme is 'making connections', that is, recognizing the common aspects of problems and solution methods both within and beyond programming. The findings and discussion will be related to the following evaluation criteria (see figure 4): (3) – higher order thinking skills; (4) – metacognition and learning to learn; (5) – reflection on problem solving processes and strategies; and (6) – near transfer. This completes the presentation of findings.

The three review tasks were designed to complement and extend the other course activities. They encompass individual and group activities in both verbal and written mode. They are analysed qualitatively in order to demonstrate the range and diversity of responses. The first review task is open-ended. Analysis of the responses to this task reveals pupils' spontaneous references to problem solving processes and strategies in relation to particular programming problems. The second task is directed towards exploring pupils' strategic knowledge; a sample of pupils was asked to discuss in individual interviews a range of problem solving processes and strategies that they had learned in unit 1. Other pupils were brought into this discussion in the later interviews (phases 2 and 3), by which time pupils had gained a wider experience of programming. These discussions focused mainly on three aspects; planning strategies (within and beyond programming); testing and debugging strategies (within and beyond programming); and contrasting alternative solutions (extending to where the alternative was generated using a different software tool). The third review task is again open-ended, and, as the last activity on the course, it enabled pupils to summate their insights on learning to program. The practical problem solving tasks on spreadsheets were drawn from the assessment portfolio and comprised a coursework assessment in two parts (core and extension) and an SEB project.

The order of presentation is as follows: the first review task – outline, findings and discussion (section 8.2), and similarly the second review task (section 8.3); further evidence from phase 2 and phase 3 interviews on planning strategies

'PRIZE' [TOPIC 10]

[PAULA]

This program was about debugging. I had to find all of the bugs and correct them carefully making sure that I did not put in any of my own.

* * * * * * * * * * * * *

I enjoyed doing this program because it did not take too long to do, but I did have to be careful that the things I was changing were wrong and needed to be corrected.

* * * * * * * * * * * * *

The most important thing that I have learned from this is how important it is to check everything, for example, that variable names are the same etc.

[CHRIS]

1. *Find the bugs in the program.*
2. *It was the first debugging I had done.*
3. *I learned to debug.*

[SCOTT]

In this task I had to correct the program.
I liked this task because I liked correcting programs.
In this task I learned how to debug a program.

Figure 34: Review task 1 – Paula's, Chris's and Scott's reviews of 'prize' (topic 10).

(section 8.4) and testing and debugging strategies (section 8.5); contrasting alternative solutions and near transfer (section 8.6); the third review task (section 8.7); review of the chapter (section 8.8); and conclusions (section 8.9). This completes the presentation of findings.

8.2 EVIDENCE FROM REVIEW TASK 1

8.2.1 Outline of review task 1

As indicated in chapter 4, this task was designed to be completed by pupils individually on reaching the end of a unit. It involved the pupil in identifying the program(s) that he or she liked best, obtaining a printed listing and sample run, and answering the following three questions: What had you to do in this problem? Why did you like this problem? What did you learn from this problem? Pupils were asked to do the written part of the task at home. There are two main focuses when examining the evidence from this review task. Firstly, how effective is the task at eliciting good responses from pupils? Secondly, what did pupils learn about problem solving processes and strategies through working on these problems?

8.2.2 Findings from review task 1

Pupils' responses reflect the main themes that emerged from the analysis of responses to item two in the questionnaires (see 7.2.5), namely that pupils liked particular problems mainly because they offered opportunities to think independently and use your imagination, to experience a sense of achievement and satisfaction, to learn and practise new techniques, or to produce useful, realistic and relevant programs.

Not all pupils did this review for all the units they had finished (there were fewer responses for later units), or responded fully to the questions. This latter point is illustrated by contrasting Paula's review of 'prize' (see 5.3.7 for an analysis of coursework in relation to this task), with Chris's and Scott's (see figure 34).

It seems that neither of the boys has exerted much mental effort to come up with these responses whereas Paula has provided some detail and explanation. Scott was unable to explain to me why he liked the task when I asked him again at interview ('I don't know, I just liked it.'), but he did appear to have learned a systematic approach to debugging his programs, according to the method that he described verbally to me – 'Get a listing of the incorrect program then go through it...Just to see, see if you can see anything – if it sticks

PRIZE' [TOPIC 10]

I liked the prize draw for many of the same reasons I liked the posters. I also liked the way each ticket was numbered.

'POSTERS' [TOPIC 9]

I liked the poster because I feel that you could use it, it's user friendly and it was fun to do. I feel that I did learn something while doing this, and it was worthwhile.

Figure 35: Review task 1 – Claudia's reviews of 'prize' (topic 10) and 'posters' (topic 9).

'PRIZE' [TOPIC 10]

1. *In this program I had to design and write a program to output prize draw tickets.*
2. *I liked this program because it was quick and easy when I used a loop.*
3. *I learned how to use a loop.*

'PATT1' [TOPIC 12]

1. *I had to design and write a program to output a line of stars.*
2. *I liked it because it was a short program and I enjoyed making the patterns.*
3. *I learned how to make different patterns using [with] stars, using procedures.*

Figure 36: Review task 1 – Lorraine's reviews of 'prize' (topic 10) and 'patt1' (topic 12).

'POSTERS' [TOPIC 9]

For this I had to design a poster, write out procedures in both the design and the program.

\+ + + + + + + + + + + + +

I liked doing this because I had to make up the poster and what it would be about, and I did the design for the procedures first which made it easier.

\+ + + + + + + + + + + + +

From doing these tasks I have understood a lot more about designing and programming[,] and using a procedure variable [sic] and variables for entering the information later on when the program is run.

'SURVEY8' [VERS. 8 OF THE PROGRAM, TOPIC 13] – RAINFALL

To get this program I continually changed my original program, each time improving it slightly. I had to output the amount of rainfall each day in November, give a monthly and an average rainfall.

~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

I liked the program because it can be used practically, in real situations. You also had to keep thinking of ways to improve it.

~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

[I learned] How to set out a table properly, I used data statements and a variable [fixed] loop, I learned how to calculate and output totals and an average, and I used tab stops.

Figure 37: Review task 1 – Paula's reviews of 'posters' (topic 9) and 'rainfall survey' (topic 13).

out, it's wrong.'[52] Chris also discussed his approach to debugging the program. He indicated that he had begun by scutinizing the coding to identify any obvious errors (just as Scott had done), then he ran the program to see which other errors were thrown up by the computer. He had some difficulty with remembering to insert the '$' symbol – 'I finally found it [that the '$' symbol was missing for some variables], but it took me about five minutes,' and with discovering the technique to number the tickets consecutively – 'There was one that I asked the teacher if that was a bug or not...It was the one to do with the loop, like it would be [the] amount of tickets, if you wanted five tickets, it would be, 'Ticket number 5', 'Ticket number 5'.'[53] From the discussion with Chris it is apparent that he did learn several things that he hadn't encountered or fully understood previously. From these findings I conclude that Chris and Scott *are* capable of reviewing their learning (and there are other findings to substantiate this), but that the review task was not a very effective stimulus for them.

Some of the other responses from pupils were uneven or superficial, for example, Claudia gave reasons for liking 'prize' (and the other problem, 'posters', which she had chosen to write about), but she did not reflect at all on what she had learned (figure 35).

Also Lorraine fell behind with her reviews and then she set about catching up quickly, and as a result some of her recollections of problems were inaccurate, for example, her description of the 'prize' problem. She mentions in each review only one technique that she has learned (figure 36).

Where pupils have responded more fully, the responses are just as likely to have focused on problem solving and metacognitive strategies as on programming concepts and techniques. Paula's review of 'prize' (see figure 34) exemplifies this – she has learned, '..how important it is to check everything.' In her reviews of 'posters' and 'rainfall survey' (figure 37) she mentions design skills ('...I did the design for the procedures first which made it easier.') and evaluation skills ('You also had to keep thinking of ways to improve it.').

Caroline's reviews of 'posters' and 'address cards' also focus on design skills and debugging (figure 38 overleaf). She draws frequently on her experiences of solving programming problems by comparing and contrasting problems, her

---

52 Interview 10, Carol-Anne, Bryan & Scott, 14/6/94, Transcripts, p.93, 105.

53 Interview 6, Chris and Lorraine, 22/2/94, Transcripts p.48-9.

'POSTERS' [TOPIC 9]

My task was to design and write a program which would display 3 printed posters. It had to include prompts for the user to input the poster details.

I enjoyed working on this program mainly because it required a lot of thinking and testing. It took me a bit longer than some other programs I have previously written but was a lot more interesting. Every time I ran the program I would discover another bug but eventually I managed to correct them all and get the program working.

I learned that [to] have a detailed design to work from really does help, and I think that had I not used a design I would have had a lot more errors. This program also gave me practice in using 'string' and helped me understand it better.

'ADDRESS CARDS' [TOPIC 10]

My task was to design and write a program to output printed address cards. It had to prompt the user to enter the address details and the number of cards required.

Again, I liked this program because it was challenging. It took even longer to complete than 'POSTER' did. I used a TOP-DOWN DESIGN which I felt was easier and more interesting to use. It helped me to set out the commands neatly and cut down on bugs in the finished program. It took a lot of work but was really interesting to do.

In this program, I learned to use TOP-DOWN DESIGNS and also to use a variable name after 'FOR counter:= 1 to ...' (this allowed the user to choose the number of cards to be printed).

I enjoyed working on this program and seeing it work in the end as it was supposed to.

Figure 38: Review task 1 – Caroline's reviews of 'posters' (topic 9) and 'address cards' (topic 10).

'DISPLAY TWO SHIPS' [TOPIC 4]

1) *In this problem, I had to write a design and design my ship on a grid in the answer booklet, using characters from the keyboard. I then had to write a procedure and main program to display two ships.*
2) *I liked this program because I designed and wrote everything myself, I felt that it was really MY [underlined twice] work. I would not have felt this if the program had been in the book and I had only copied it.*
3) *I learned more about the use of procedures in programming. I felt it was easier to understand this way rather than just reading about it.*

'DISPLAY TWO ROBOTS' [TOPIC 5]

1) *This was actually an assessment task. My task was to design and write a program (using a procedure) which would display two robots on the screen. As with the ships prog., I then had to obtain a printed listing and a sample run.*
2) *As before, I liked the program because I designed it all myself, I enjoyed drawing the robot and seeing it appear on the screen. I also liked the fact that it appeared in colour on the screen (unlike ships).*
3) *I didn't really learn anything new from this program but I did get some practice in using procedures and colour.*

Figure 39: Review task 1 – Caroline's reviews of 'ships' (topic 4) and 'robots' (topic 5).

progress with solving them and her reactions to them, e.g. 'Again, I liked this program because it was challenging. It took even longer to complete than "POSTER" did.' This is also evident from her earlier reviews of unit 1 problems, 'ships' and 'robots' (figure 39) – 'As with the ships program...', '...it appeared in colour ... (unlike ships).', 'As before, I liked...'. She is clearly very analytical in her approach.

Her reviews are interesting because they reveal other things about her metacognitive insights and her approach to learning, for example: she knows that she will understand better through applying a new technique, rather than, '...just reading about it'; she knows that she will gain more satisfaction from the work if she is given scope to use her own ideas and to find her own solutions; she distinguishes between situations where she is learning a technique for the first time, and practising it (but nevertheless she seems to attach importance to the latter); she appraises the techniques that she is learning (e.g. 'It [top-down design] helped me to set out the commands neatly and cut down on bugs in the finished program.') rather than applying them unthinkingly; she actively seeks out connections, without prompting; and she knows that effort and perseverance are important for success. It is hardly surprising that she does succeed so well with her programming.

8.2.3 Discussion of the findings from review task 1

Review task 1 was a very good discriminator to reveal the extent of pupils' willingness and/or ability to reflect, in a written form, on what they have learned, and this is useful for the purposes of this research. It could (and perhaps ought to) have been used more formatively to assist all pupils to reflect carefully and fully, and to set out their thinking clearly in writing, so that the gulf between the 'good' responses (such as Caroline's) and the 'poor' responses (such as Chris's) narrowed as pupils produced their reviews for the later units. Also more frequent practice at reviewing solutions could have resulted in greater improvements. One would have to be very careful, however, about the nature of any intervention to prompt fuller responses. It is, after all, a personal response that is being sought from pupils, and direct teaching could produce a coaching effect which would result in pupils giving stock answers that didn't reflect their thinking.

I suspect that the brevity of some responses is an indication of the pupil's reluctance to do the review task at all. Kafai (1993) got her students, who admittedly were much younger, to keep a 'designer notebook', in which they made an entry each day. Its purpose was two-fold: to prompt students to think

Review Task 2

As well as discussing your answers with me from the Unit 1 questionnaire I also want to ask you these questions about problem solving.

Could you please think over what you will say. You may also want to write down some notes. It isn't a test, so don't worry if you are not sure about your answers.

1. Can you show me somewhere, in Unit 1, where you had to...

   (a) starting with a new problem, break it down into a number of smaller problems?

   (b) take one of the steps in the design and work it out in more detail?

   (c) write a program, basing it on a design that you've been given, or on your own design?

   (d) test your program to see that it does what it is supposed to do?

   (e) find and correct the bugs (mistakes) in a program?

   (f) having written a program, amend it to do something different?

   (g) having written and tested a program, suggest ways of improving it?

   (h) compare two different solutions to the same problem?

2. In general, do you feel that the strategies that you have been taught for solving programming problems help you to become more efficient at programming?

3. Are you aware of tackling problems in other subjects or outside school in similar ways?

4. Do you think there is anything that you've learned in Unit 1 that might help you to learn other things?

Figure 40: Review task 2.

about their plans, problems and changes; and as a tool for data collection. She notes: 'At the students' request the notebook writing did not continue after the summer break.' (p.93)

Some pupils in the case study class have 'voted with their feet' by giving only cursory attention to the task or failing to do it at all for the later units. The reasoning behind the review task was that every pupil could find at least some tasks that he or she liked in each unit, and that this would provide a positive focus for reflection. But perhaps some pupils did not like any of the tasks in a unit, or they may have felt that they didn't learn very much from them? From responses to the questionnaires and during interviews, this seems an unlikely explanation. A more plausible explanation is that pupils had other homework and extended coursework to do that they considered to be more pressing and/or they did not see the point of the exercise.

Experiences from using this review task indicate that it is very important not to rely on written reviews as the *only* means of getting pupils to reflect on their learning. If it becomes just another classroom routine, a demand on pupils' time, then it will not be effective. All pupils need to understand *why* they are being asked to review their learning, and so teachers must discuss a thinking skills and learning skills agenda with their pupils. Also some pupils do cope much better with verbal questioning (such as Chris and Scott), perhaps because they are not good at self-prompting, and they would need to be taught the skills of organizing their thinking and structuring their written responses accordingly.

8.3 EVIDENCE FROM REVIEW TASK 2

8.3.1 Outline of review task 2

The second review task (figure 40) relates to the problem solving processes and strategies that were taught in unit 1 (the preamble to the task, although it doesn't state so specifically, refers to the first phase of interviews, in which Chris, Lorraine, Paula, Claudia and Scott participated; pupils were briefed individually about the task approximately one week before each interview was scheduled). The responses to this task are verbal. Other pupils were brought into this discussion through questioning in the later interviews.

The question of which pupils to interview following completion of unit 1 was not crucial to the research design, since this phase of interviews was exploratory and the issues raised would be revisited in later interviews (phase

2 and phase 3) to be conducted with all pupils. The choice of interviewees did however reflect the gender balance in the class and encompassed pupils who responded differently to the questionnaire, indicating a range of initial reactions to the learning environment.

8.3.2 Findings from review task 2

The main findings to emerge from this review task are: (1) all five pupils could identify, mostly without prompting, where each process came into play in unit 1; (2) pupils could reflect on these processes, for example, on how they were able to refine steps or debug solutions, and on whether they considered that the processes were worthwhile to carry out; (3) the strategy of breaking a problem down into smaller parts was viewed as applicable to non-programming contexts by Paula and Claudia; (4) ideas on transfer of learning from programming to new situations were still welded to the context of using computers in classrooms.

The account below from Claudia's interview (not a verbatim one, but taken from my notes and verified by her) illustrates the types of follow-up questions that I asked, which varied slightly from pupil to pupil according to their responses.

MK: *[Item 1]*
Can you show me somewhere, in Unit 1, where you had to...

MK: *[1a]...starting with a new problem, break it down into a number of smaller problems?*

CL: Topic 3 Task 2 [arrange given steps in order to display your name and address on the screen].

MK: *[probe] Do you find that this helps you to solve the problem?*

CL: Yes, because the design is in English and you can talk about it [the problem].

MK: *[1b]...take one of the steps in the design and work it out in more detail?*

CL: Topic 5, the procedure to display the ship.

MK: *[1c]...write a program, basing it on a design that you've been given, or on your own design?*

CL: The robot [Topic 5 Task 10]. You had an idea of what you had to do, but you had to do everything yourself.

MK: *[1d]...test your program to see that it does what it's supposed to do?*

CL: You have to test all of your programs, especially Dennis [Topic 4 Tasks 5 & 7]. I don't think anybody could get it right first go! I was a bit baffled by it, but it didn't put me off!

MK: *[1e]...find and correct the bugs (mistakes) in a program?*

CL: Debugging is something you do all the time.

MK: *[probe] How do you go about doing this?*

CL: You check the program listing and it's usually quite obvious. I print out the longer ones.

MK: *[1f]...having written a program, amend it to do something different?*

CL: [Points to problem in Topic 5 where, instead of the output consisting of four 'vans' as occurred in the previous solution, the output has now to consist of three 'vans' in different colours.]

MK: *[probe] Why [are you directed in the materials to] amend the design first before changing the program?*

CL: You made changes to the design first so that you really knew what you were going to do, and so that you didn't get all confused.

MK: *[1g]...having written and tested a program, suggest ways of improving it?*

MK: *[prompt] Think of the two assessment tasks, joke and robot.*

CL: You could move the robot into the centre of the screen rather than have it on the left. There wasn't really much that you could improve.

MK: *[probe] Do you think that it is worthwhile to do this [suggest improvements]?*

CL: Yes. If you encountered the same problem, or a similar problem again, you would know how you could do it better.

MK: *[1h]...compare two different solutions to the same problem?*

MK: *[suggestion] Lets look at the 'vans' problem in Topic 5.*
[probe] What kinds of things might you compare?

CL: A stranger would find the teacher's version [the one without procedures] easier, reading it from top to bottom. I prefer the other version because I understand it, its shorter and easier to type in.

MK: *[probe] Do you think that it is worthwhile to do this [make comparisons]?*

CL: Yes, its better if you're shown [the differences between them], you would know better in future which version to use. Up to that point [when you were asked to compare the two programs], you knew about the procedure and how to use it, but you didn't know <u>why</u> you were using it.

MK: *[Item 2]*
In general, do you feel that the strategies that you have been taught for solving programming problems help you to become more efficient at programming?

CL: Yes. You wouldn't have come up with them yourself, unless it was just a coincidence. And if you had come up with them, you would hardly notice that you were using them.

MK: *[probe] Why do [i.e. is it that] you think it is important to be aware that you are using a particular strategy?*

CL: It isn't good if you don't know what or why you're doing something, or the consequences of an action.

MK: *[Item 3]*
Are you aware of tackling problems in other subject areas in similar ways?

CL: You probably are. You do break problems down, in and out of school, without thinking about it.

MK: *[probe] Can you give me a specific example?*

CL: Computer games. You have to think about each level.

MK: *[Item 4]*
Do you think there is anything that you've learned in Unit 1 that might help you to learn other things?

CL: Yes, when I go on to do more programming.

MK: *[prompt] Apart from programming?*

CL: There probably is, but I can't think of anything.

[Interview 4, Claudia, 9/11/93 & 23/11/93, Transcripts p.29–31]

Overall Claudia demonstrates a good understanding of problem solving processes in programming, even although she has spent only a short time learning programming prior to this interview. Furthermore she values the explicit emphasis that is placed on problem solving processes in the course because it aids her understanding and makes her more efficient at programming.

The noteworthy aspects of Chris's, Lorraine's, Paula's and Scott's responses are as follows[54]:

- Item 1a

Breaking a problem down into a number of smaller problems was linked to the program design process (i.e. creating an ordered list of steps to describe the solution) and also to the idea of making a plan before embarking on the solution, in accordance with the way that pupils had been taught to do this in programming i.e. first you read and analyse the problem, then you design the program etc.

---

54 Interview 1, Chris, 15/10/93, Transcripts p.1-10; Interview 2, Lorraine, 15/10/93, Transcripts p.11-7; Interview 3, Paula, 1/11/93, Transcripts p.18-27; Interview 5, Scott, 29/11/93 & 30/11/93, Transcripts p.32-41.

Did the strategy provide any leverage on problems? Managing complexity was an underlying theme in both Lorraine's and Paula's responses.

> ***Lorraine:*** *...because if you break it [the problem] down you don't have to do it all at once, you can take your time, it's easier to understand it. (p.12)*

> ***Paula:*** *...because you are not faced with everything at once, you can do one thing at a time and then move on to the next thing... (p.22)*

While Paula viewed the process of producing a written design for the program as helpful ('...you know what you've to do when...it tells you to sort it [the program] out properly – in order.' p.23), Scott viewed it as unnecessary and time-wasting. He suggested, 'It would have been better if you took out writing it in English because you've got it on the design grid.' (p.36)

Chris mentioned a difficulty with designing programs, which was that he tended to, 'leave things out,' and when asked to think of ways of overcoming this problem, he suggested, 'Look at the problem description and see what it [the program] had to do and make sure the design had it in it.' (p.9)

- Item 1d

There was agreement with Claudia's observation that testing is something you do with all of your programs; you run the program and check it – both its operation and the listing (none of the programs in unit 1 involved any inputs and therefore choice of test data was not an issue).

- Item 1g

Chris, Lorraine, Paula and Scott agreed with Claudia that suggesting improvements to a solution is worthwhile.

> ***Chris:*** *In case you're doing a later one. You could always look back... (p.8)*

> ***Lorraine:*** *It saves like thinking for ages and ages what you could do to improve it [if you were asked to in future]. Like, you've already thought about it. (p.16)*

> ***Paula:*** *...if you do something, it's better to do it the best you can and get the best results.... You can make them [your solutions] better all the time by doing different things... (p.25)*

> ***Scott:*** *...because you might come across it again...making it look a bit more exciting (p.39)*

It is unclear from Lorraine's and Scott's responses whether they were thinking of the identical problem being encountered again, or whether transfer would be involved (with Claudia's response it is clear that she envisages either situation). Paula's response echoes closely Polya's view that, 'There remains

always something to do; with sufficient study and penetration, we could improve any solution.' (Polya, 1948, p.14)

- Item 1h

Again Chris, Lorraine, Paula and Scott agreed with Claudia that it is worthwhile to compare alternative solutions to a problem. Paula exemplified this by referring to the advantages of one particular solution over another; the program which used procedures was, in her opinion, easier to write because it was shorter, easier to amend, and easier for her to understand[55].

Chris: *So as people know which way to write out their programs... (p.7)*

Lorraine: *Probably so that you understand what was different about it... I could probably go back to it and see what was different about it. (p.14)*

Scott: *Because it helped you to think about what you were doing...It just shows you what other ways it could be done. (p.40)*

The responses suggest that comparing and contrasting solutions to a problem is an effective way to achieve a better understanding of techniques and also to achieve a better solution.

- Item 3

In response to the question, 'Are you aware of tackling problems in other subjects in similar ways?', Paula referred to and exemplified the strategy of, 'breaking a problem down into a number of smaller problems'.

P: Yes, because you get the problem and it's like you don't have to face the whole thing at once. You can break it down into small problems, and then just take the small problem instead of the whole big thing at the one time.

MK: *Can you give me any specific example of having done that – not in programming, but somewhere else?*

P: Well if you're in say English, or in different subjects at school, and you are writing an essay or something. Instead of trying to think of what you are going to write for the whole essay, you think of what you are going to write for your opening paragraph and then what you're going to write about it – and at the end how you are going to summarise what you have written, instead of just saying, 'I've got an essay to do, well I can start it like that', then just rush through it. You've got it spaced out for you in little problems, how you're going to open it, just what you are going to do in the middle, how you are going to end it.

MK: *Does your English teacher encourage you to do it in that way?*

---

55 These were the three criteria that were applied in the original task (Unit 1 task sheets, topic 5, task 6).

P: Yes.

[Interview 3, Paula, 1/11/93, Transcripts p.26]

This was, I thought, an interesting parallel to draw; I had anticipated a reference to mathematics. My surprise reflected my ignorance of how English is taught, and it occurred to me that pupils were in a far better position than secondary teachers to identify common aspects across subjects (apart from the obvious things, such as doing calculations, drawing and interpreting graphs, and literacy skills).

- Item 4

In terms of learning that can transfer to other contexts, Paula's response was similar to Claudia's except that she referred to using computers elsewhere (her example was in the maths class), rather than doing programming again in future.

8.3.3 Discussion of the findings from review task 2

Although pupils had not spent long on programming when this review task was done, it is evident that they were already building an understanding of programming and problem solving processes. Polya (1948) identifies, in the preface to *How to Solve It*, his motive for writing the book, as, 'Trying to understand not only the solution of this or that problem but also the motives and procedures of the solution, and trying to explain these motives and procedures to others.' (p.[vi]) This is precisely what pupils were engaged in doing here, and for the most part they managed to do it very well.

It is interesting that, in response to item 3, Claudia and Paula were able to identify a strategy that they considered to be generalizable, whereas item 4 did not bring anything in particular to mind for them, other than computer related activities. Perhaps the earlier questions of the review task prompted their thinking about specific strategies and cued them towards the responses that they gave, whereas there were no questions which were focused more generally on learning. Also item 3 enabled pupils to draw on past experiences of problem solving in a concrete way, whereas item 4 focused on future experiences of learning, which requires an effort of imagination and some degree of speculation.

8.4 PLANNING STRATEGIES: FURTHER EVIDENCE FROM PHASE 2 & PHASE 3 INTERVIEWS

The outcomes of the second review task led me to probe the following aspects during later interviews: (i) planning strategies (within and beyond programming); (ii) testing and debugging strategies (within and beyond programming); and (iii) contrasting alternative solutions (within programming and extending to where the alternative was generated using a different software tool). Each is discussed in turn in this section and the following two sections.

The issues which are examined under the heading of *planning strategies* are: Did pupils consider the written design process in programming to be necessary and helpful? What were the problematic aspects of applying top-down design? Was the principle of top-down design understood? How did pupils cope when they encountered difficulties with designing programs? How did pupils locate their experiences of design in programming with other experiences of design or planning processes?

8.4.1 Did pupils consider the design process in programming to be necessary and helpful?

Paula's and Claudia's theme of breaking complex problems down into smaller, more manageable, problems recurred in discussion with other pupils, both in relation to the manner of designing their programs and their overall strategy for tackling programming problems.

Having or constructing a written design for a program was viewed by many pupils as important for a variety of reasons, including: as an aid to understanding and discussing the problem and its proposed solution; eliminating bugs before the program is written; making it easier to locate bugs in a program during testing (particularly when the design is modular); as a means of documenting the solution to aid amendment at a later stage; and to aid the process of constructing new solutions by synthesizing elements of previous solutions. This is not to say that all pupils found it easy to produce written designs for their programs, as it is a stringent requirement to produce a clear, accurate, complete and unambiguous representation of the solution.

8.4.2 Which aspects of top-down design were problematic?

Chris, Stephen, Claudia and Catherine discussed with me the aspects of top-down design that were problematic for them, which I have classified under

four headings: How to set out the design in levels; Which steps should be considered as main steps?; Which steps should be refined and when to stop refining?; Figuring out the steps.

- How to set out the design in levels?

> ***Chris:*** *It was just when I was refining some of the steps, at first I got it wrong, it was like, instead of like step 6 refinement ... I was making it a further big design, which means I was making it [the top level] longer.*
>
> *[Interview 6, Chris & Lorraine, 22/2/94, Transcripts p.49]*

We saw evidence of this difficulty in Chris's design for the 'address cards' problem in topic 10 (section 5.3.6) and later in the topic 16 assessment (section 6.3.8).

- Which steps should be considered as main steps?

> S: That sometimes as well, 'leave a blank line', I didn't think of that as a main step.
>
> *MK: You might just have added that to the program later?*
>
> S: Yes.
>
> *[Interview 20, Stephen, 28/2/95, Transcripts p.252]*

- Which steps should be refined, and when to stop refining?

> ***Claudia:*** *I wasn't sure why some steps were refined and not others. I thought some of the steps didn't need to be refined and I could just have written the program without breaking it down any more.*
>
> *[Interview 8, Claudia, Kirsty & Kelly, 10/5/94, Transcripts p.66]*

- Figuring out the steps

> ***Stephen:*** *It's quite hard to figure out the refinement bits ...*
>
> ***Chris:*** *...I always miss out the wee ones like, read the whatever ... I just get the main points of it, then I have to go and change it – my program's wrong.*
>
> ***Catherine:*** *Just like whenever you are like thingmying the design, you know, how like before you write the program. Well that sometimes you think, 'Oh, what do I put in here?'*
>
> *[Interview 20, Stephen, 28/2/95, Transcripts p.253; Interview 14, Paula, Chris & Kirsty, 20/12/94, Transcripts p.165; Interview 15, Catherine & Kelly, 13/1/95, Transcripts p.190]*

The comments reflect pupils' concerns about applying top-down design consistently, the accuracy of the resulting program if the design is incomplete,

and the difficulty of identifying the solution steps (which is a problem of analysis). The design decisions that are part of this process are not straightforward ones; there is always an element of judgement involved, even for experienced programmers.

8.4.3 Was the principle of top-down design understood?

I questioned pupils on their understanding of top-down design in programming, and on the whole the principle appeared to be well understood. I shall present two cases (Paula's and Stephen's) to signal the range of responses.

Paula seemed to have the most secure grasp. I put her understanding to the test by asking her to explain to me, when she had reached the end of the course, why she felt a top-down approach made it easier to program, and how top-down and bottom-up methods compared.

P: It [top-down design] was a lot easier if you had everything all set out, like what you had to do and the order you had to do it. Then as you got to a bit and you had to refine it, you didn't have to worry about everything that you had to do right away – you know at the one time, you could break it down into the wee refinement steps. So you didn't have to deal with the full thing as you were writing it all out.

...

MK: How do you think those two techniques [top-down and bottom-up] compare?

P: I think probably the top-down would be better because you know the main steps, you are not missing out anything – like if you started from the bottom and worked up if you were doing a bit at a time. It would probably be quite good doing it a bit at a time, but you would probably miss out bits and it wouldn't be all in the order that you needed it.

...

MK: You find ... translating the design into a program fairly straightforward?

P: Yes.

MK: Why do you think that is?

P: Because it's .... like it says there, 'select mode 7', [refers to design on page 3 of Unit 3] and all that. You know it would be 'mode dot equals 7', you just know all the commands and all you need to do is change it from into English into computer language.

[Interview 23, Paula, 9/3/95, Transcripts p.317–8]

Paula's ability to translate the design into a program straightforwardly hinges upon two inter-related aspects; the problem has to be broken down in a manner which makes it amenable to computer solution, and knowledge of language constructs and syntax (the factor that she has identified) organized as

plans or templates. Conversance with language constructs and plans facilitates the process of breaking a problem down appropriately, leading also to a more efficient solution since elements of previous solutions can be re-used. Additionally, the models of the design process within the task sheets provide examples and guidance on how this can be done. To develop an effective solution strategy in general requires knowledge of the tools and materials at one's disposal, and modelling provides a means to demonstrate how this knowledge can be brought to bear on a particular solution. Providing useful models for beginners can help to prop up their rather shaky knowledge.

Stephen was one of the most hesitant in response to questions, but nevertheless he did appreciate that as he went down a level there was a more detailed breakdown of the problem.

MK: *What is meant by the design?*

S: I don't know – the steps of it, or something.

MK: *Okay, that's the design there, [pointing to bottom half of page 45] so what would you say that was?*

S: A plan?

MK: *It's a sort of plan for the program, isn't it?... Why do we have what is called a top level here, and then refinements?*

S: Because you have got the in between bits.

MK: *What's an, 'in between bit'? Do you think we have got missing steps here?*

S: Yes, all of that is just a rough bit [points to the top-level of the design] and that's the detailed bit of it [refinements below].

MK: *Just to be clear here, is this [the top level] – does this have missing steps? – is it incomplete?, or is it just not detailed?*

S: It's not detailed.

MK: *Right, so that's the top level then, it's, if you like, the <u>main</u> steps to obtain the solution, and when you start to work them out in detail you then <u>refine</u> the steps.*

S: That sometimes as well, 'leave a blank line', I didn't think of that as a main step.

MK: *Is that something that you would have just forgotten about and didn't think was that important?*

S: Yes.

MK: *You might just have added that to the program later?*

S: Yes.

MK: *I would agree that it wouldn't be a major omission, but if you forgot [the step] to output the poster, then that would be a major omission.*

S: Yes.

[Interview 20, Stephen, 28/2/95, Transcripts p.252–3]

8.4.4 How did pupils cope with difficulties?

Pupils coped with difficulties in various ways, according to interview responses. Some common strategies emerged and these are exemplified by Catherine's, Chris's, Stephen's and Caroline's comments. Further evidence was needed in David's case to pin down his actions more clearly (this evidence has been drawn from field notes).

- A common strategy was to adopt a 'bottom up' approach:

CT: Well that sometimes [when you are working on the design] you think, 'Oh, what do I put in here?'

MK: And how do you resolve that problem...?

CT: Do the program.

MK: Do the program, you go back to write the design, that's interesting.

[Interview 15, Catherine & Kelly, 13/1/95, Transcripts p.190]

and:

CH: ...I always miss out the wee ones [in the design] like, read the whatever ... I just get the main points of it, then I have to go and change it – my program's wrong.

MK: Do you think you would probably find it easier to go straight to coding without having any design stage first?

CH: It's easier to do that, I've done that.

MK: You have done that. When did you do that?

CH: I've done it in Topic 17, I missed out the algorithm then went back to it.

MK: Is there any value in writing it up later?

CH: No!

MK: Why do you think it is easier for you to go straight to coding it, Chris?

CH: I don't know, cause it's better working with the commands and that for the program than writing it in English...

[Interview 14, Paula, Chris & Kirsty, 20/12/94, Transcripts p.165–6]

Writing out the design afterwards could have served as an additional check on the logic of the program, but it didn't occur to Chris to use the design in this way.

- Another strategy was to edit a previous program:

  MK: Do you find that writing the design out first does help you to write the program?

  S: It depends what one it's on. Some of the programs you can see from the other pages and just copy it back out.

  [Interview 20, Stephen, 28/2/95, Transcripts p.256]

- A further strategy, described by Caroline, was to work backwards and forwards interspersing designing with implementing. This is akin to the way expert programmers obtain solutions (Linn, Sloane and Clancy, 1987).

  MK: Are you able to design your programs before you go to the stage of coding them?

  CI: Usually yes. Sometimes it [the design process or the written version of the program?] doesn't work but ....

  MK: What do you do in that situation then? Do you work on at the computer?

  CI: Yes, I go to the computer then I change the [written] program at the end once it works on the computer.

  MK: Do you ever reach a stage when you are designing the program where you want to leave it without finishing it and go straight onto the computer?

  CI: I've done that before, yes. Then I can see what I've to do next, then put that down as the next step.

  ...

  MK: At least when you are testing your program at the computer you get some feedback immediately.

  CI: Yes, and that can tell you what you have done so far is right or not, saves messing up the whole thing.

  ...

  MK: Okay. But you still find the designs helpful?

  CI: Yes, because it gives you something to work from.

  MK: Do you find that if you are updating a design from a previous program that you are able to carry out that process quite smoothly?

  CI: Yes, because that is just giving you the foundation to write it and you are just adding in your own bits.

  MK: Are the hardest designs to write the ones that you are really writing ....

CI: Starting from scratch. I really need something to look back, a similar problem to look back at, because I'm not very good at writing them just straight from scratch.[56]

[Interview 25, Caroline, 20/3/95, Transcripts p.359–61]

Caroline is sometimes feeling her way forward, rather than following an exact plan, which fits with Papert's (1994) description of the bricoleur, however it is not her preferred way of working. In the later topics pupils generally found that the design process became more testing because the problems were less analogous to each other.

- Sometimes it was very difficult to discern any strategy.

I was unable to discern, from the interview with David, how he overcame difficulties with designing programs in the later topics. When he had finished unit 2 his view on top-down design was very close to Scott's, that it provided little leverage on problems:

MK: The idea of listing the main steps of the design first and then refining some of them, that's called top down design. Are you confident about [doing] that?

DA: Yes.

MK: ...Do you find that it is useful to design them before you write the program?

DA: Sometimes it takes too long to sit and write them down.

MK: Could you write the program without having first written the design?

DA: Yes. Maybe one or two mistakes, but it would be easy to fix.

MK: Do you think that would always be the case, or is it just if the problem is simple?

DA: Most of the time it would be the case.

[Interview 11, Dawn & David, 20/6/94, Transcripts p.125–6]

However by the end of unit 3, his view had altered in the sense that he was now finding top-down design to be difficult rather than straightforward to apply.

MK: What do you find is the hardest part, David?

DA: Writing out the designs.

MK: Do you find it easier to write the program than to write the design?

DA: [Yes] Because you don't follow [understand] the design.

---

56 This comment is also quoted earlier in 7.2.8 when I discussed pupils' reasons for feeling confident or unsure about particular aspects of the work.

[Interview 21, David and Claire, 6/3/95, Transcripts p.274]

I found David's explanations unclear, for example, when asked why he didn't always appear to base his programs on his designs (see the earlier references to his work in sections 5.3.4, 5.3.6 and 6.3.7), he replied:

> DA: I just think you write it [the design? the program?] and you decide that something would be better if it was, I don't know [?]. I don't always follow [understand] the programs when I'm writing them out.
>
> *[ibid., p.275]*

I decided to try a different tack, by questioning David about his understanding of a topic 14 design that was given in the notes (Claire also contributed to the discussion).

> *MK: As an example, and drawing on what you have been doing recently, if you did have, say, within your design, a step that said, 'repeat', and then another step further down that said, 'until such and such a condition was met', what do you understand by that?*
>
> C: It just means that it's going to keep on going until whatever it's doing – it goes how you want it.
>
> *MK: Until the condition is met, Claire?*
>
> C: Yes.
>
> *MK: When you say it's going to keep on doing whatever, what is it going to keep on doing, which part?*
>
> C: [Unclear, background noise]
>
> *MK: Would it help if we looked at an actual design?*
>
> DA: Steps in the loop.
>
> *MK: Here is one on page 44, 'step 4 – repeat', and there are three steps here inside this loop, '5 – find the result of one dice throw', '6 – output the result', '7 – add one to the number of rows', 8 is 'until result equals 6'.*
>
> *So if you were reading that, you would understand that these three steps would be repeated until you got a result which equals six?*
>
> C: Yes.
>
> *MK: What else is happening in that program?*
>
> C: Adding.
>
> *MK: Yes. What is it adding up?*
>
> DA: The number of throws.
>
> *MK: That's right, David. Because we start off with a counter which we set to zero and then each time we throw a dice we are adding one to the counter. Once we've got a result of 6 we can output the value of the counter, the number of throws, okay?*

Field notes – 10/2/95

David: *Having lots of problems with his program [Topic 14 Task 10]. He does have a written design – adapted form the previous one – which looks correct, but he has started to make haphazard changes to the program when he found it [the program] didn't work... He had within the 'display_title' procedure the output of each variable (result1, result2) which had not at that stage been assigned values...*

[Intended] Follow-up: Can we print out David's buggy program? Work with David on problem.

Field notes – 13/2/95 (after school class)

David: *Continuing with [Topic 14 Task 10]. We started [working from the written design] with the positioning of headings & used grid to determine column positions. We traced program establishing that it would work correctly up to the point of generating each result randomly. The mention of the word 'randomly' helped David to recognize incorrect syntax (he had result1 (1,6) instead of result1:= RND(1,6)). This resolved problems with the program... We discussed the next problem [Task 11] and David suggested 'until result1= 6 and result2= 6' and I urged him to try it. He looked very pleased when it worked!... David was referring to the design and had it opened alongside the printed listing...A good outcome for him today.*

Figure 41: Extract from field notes of 10/2/95 and 13/2/95 – David's design difficulties.

C: Yes.

MK: What would happen if you had step 3 in the wrong place? Say you had it coming after [step] 4?

DA: ...

[ibid., p.276–7]

David seemed to understand this design quite well. Perhaps this was one of the easier designs to understand. One clue about why there were mismatches between his designs and programs was when I observed his attempts to write a topic 14 program (task 10) without seeming to refer to the design that he had just written. The notion that the design should serve as a blueprint for the program had either not occurred to him, or, more likely, he had abandoned his design early on once he discovered that he couldn't get his program to work (figure 41).

David had been very easily knocked off course when incorrect syntax resulted in his program not working. He deviated from the design and introduced further errors which compounded his problems. This would seem to be a very good instance of, 'strategic shortfall exacerbating the fragile knowledge problem,' (Perkins and Martin, 1986, p.226).

Linn and Clancy (1992, p.125) describe computer programming as a 'messy' domain, and they blame texts and teachers for presenting an unrealistic model of design processes in programming which, they argue, impedes students' acquisition of design skills.

> texts frequently feature what is called 'top-down design,' the process of designing a program by breaking the high-level statement of the problem into parts and then continuing with this process until the program is completed ... in reality, competent programmers engage in top-down, bottom-up, and middle-out design as well as lots of backtracking and revision of their initial ideas. Instruction that suggests program design proceeds in an uncomplicated, top-down fashion confuses and frustrates students.

This is not, of course, an argument for failing to provide any instruction on top-down design (which would add to the strategic shortfall), or for allowing learners to sort their own way out of the mess time after time. However a distinction has to be made between the competent programmer (referred to above) and the novice, such as David. It can be difficult to present, in a written form or verbally, a discussion on a range of possible solution methods which is neither long-winded nor confusing to the beginner, and it may be sensible to

concentrate on one method for a time, until it is understood (as occurs in the task sheets). There is no set pattern to the manner in which top-down, bottom-up, middle-out design, backtracking and revision are invoked by programmers in the course of obtaining a solution; it will depend on the particular impasses that are reached during this process and on the outcomes of on-going monitoring and checking. The natural points of intervention for the teacher to guide solution attempts are as the pupil works on a problem, and also afterwards, as the pupil reviews the solution. In this way pupils learn to recognize that all of these methods play a legitimate role in the solution process.

8.4.5 Generalizing planning and top-down design beyond programming

How did pupils locate their experiences of planning and top-down design in programming with other experiences of design or planning processes? The findings below have been selected to indicate the range and contrasting nature of responses. The main cases which are discussed are Paula, Stephen, David, Caroline and Claire.

In the second review task Paula gave the example of planning an essay in English as an illustration of a situation where she would sub-divide a problem. When prompted to think of other situations which involved a planning process, Stephen and David also gave this as an example, but Stephen didn't see it as 'like in programming'.

MK: *You said to me it [top-down design] was like planning something, have you ever set about planning anything else in that kind of way?*

S: Just my English, but it's not like that.

MK: *Tell me what you do in English then.*

S: Just write out a plan with wee paragraphs, underlining the important bits.

MK: *Right, so it's a paragraph plan then for, what is it you are writing?*

S: Whatever, an essay. Just like the TRIS model thing.

MK: *What?*

S: TRIS model.

MK: *Is this a way for writing essays that you have been taught?*

S: Yes.

MK: *Explain it to me then.*

S: Yes, first one is themes, no, title then themes, then the 'R' is for restate the themes again, 'I' is illustrations, then 'S' is summary.

MK: Right, so that's an outline for producing an essay.

S: Yes. We use it for everything that we do. We have to write everything with that plan.

MK: ...did you mention something about underlining?

S: Yes, in the TRIS model you underline the important bits of it, of any plan that we use, so that when we come to write in it, we know the important bits to write in.

MK: Right, I see. That sounds quite sensible to me, do you find it helps you in English?

S: It helps in English.

MK: In what way does it help?

S: It means I'm not just sitting writing, jumping from one bit to another all the time. It's in order, so everything's ....

MK: Sequenced?

S: Yes.

MK: Logical?

S: Yes.

MK: Do you think that this technique here, [points to design on page 45], – can you see similarities then? What would be the equivalent of refining in terms of your English essay?

S: Underlining it.

[Interview 20, Stephen, 28/2/95, p.253–4]

In one respect, Stephen's illustration of using a model to guide the writing process was unlike doing a top-down design for a program, since the given structure was common to all writing tasks but all programs do not have an identical structure – the nearest equivalent would be programs which conform to an input – process – output model. But in another respect it was similar, in terms of there being a systematic and hierarchical approach to the design of the essay with the key points (or main steps of the argument) being underlined in each paragraph.

David's response differed from Paula's and Stephen's because he did not see any value in planning essays before writing them.

MK: What kind of things do you plan in real life, not necessarily by writing down a plan but by thinking it out? When do you have to plan?

DA: Don't know.

MK: *It's not something you are very aware of doing. What about in some of your other school subjects, are there any times when you are aware – perhaps [of] your teacher saying, 'Plan this first before you go ahead and do it.'*

DA: English, before you wrote an essay you would sometimes make a plan of it, which basically is what it was going to be.

MK: *Okay, so what would a plan for an essay look like then?*

DA: It's like, you put your introduction, first paragraph, second paragraph, Introduction, complication,.... solution.

MK: *Does that help, do you find it helps you to write better essays?*

DA: No, I hate planning essays before I write them. I always write them better, well not really better, but it doesn't make any difference if I plan it.

MK: *It doesn't work out better?*

DA: No. Doing a first draft as well, that doesn't help.

MK: *So you find you do just as well if you sit down and just write the thing from beginning to end?*

DA: If I know what I'm doing.

MK: *How can you be sure that you know what you are doing? Give me an example of a situation where you would know what you are doing?*

DA: Obviously if it was [about] a novel [that I'd read] you would obviously wouldn't have to do a first draft, because you know all there is to know about it already.

MK: *What if you were asked to write about something from your imagination? Would you be able to do that without planning?*

DA: Yes, I don't know why I just always seem to....

MK: *That's just your way of working?*

DA: Yes.

[Interview 11, Dawn & David, 20/6/94, p.127]

David's comment about, 'knowing all there is to know' about a situation being a reason for not needing a plan or first draft is illuminating in terms of his view of design in programming. One can see why some of the problems in the earlier units, which are simple to explain and unambiguous, would not stimulate him into any planning activity. But by not heeding the planning process in his earlier work, he has lost out on the opportunity to build his skills gradually.

Caroline discussed with me in a fairly detailed manner her experiences of problem solving in mathematics and chemistry in order to explain to me why she regarded herself as, 'really not good', at problem solving. She began by

expanding on her observation on problem solving in programming, 'I really need... a similar problem to look back at.'

CI: I just think that when I get a problem, even if I understand what they mean I don't know what to use to solve it. So like I need to have other things. If I didn't have other things that were similar to work from then I wouldn't be able to do it at all, because I can't find a solution in myself, I've got to use other things to get it.

I asked Caroline, was this a problem that she had in other school subjects? Could she exemplify?

CI: Yes, in maths if they give you a problem just with a diagram and they give you the question underneath it and you've got to think of a rule or something that you've learned to solve it – I can never think what it would be. I would have to be told, 'use whatever rule to get this', because I just can't think looking at the question what I would use.

MK: Right, so you can't recognise what type of problem it is?

CI: No, not unless I'm told or unless it's really clear and I've just done something like that.... But in an exam it's all mixed up and I never know what to do.

MK: Does your maths teacher emphasise being able to recognise what type of problem it is – is that emphasised in the teaching?

CI: Well not really no, it's just now that we are doing past paper booklets that we are having to learn how to recognise it now. Not really, at the time you just do everything according to what you are learning at the time. Never get any practice of it [recognizing problem types]

So what did she do if she didn't recognize how to solve a problem?

CI: If like we were given a triangle with different sides marked or angles or whatever then I might think, is it trigonometry you have to use? or is it the cosine rule?, or is it whatever? If it's a complicated diagram, like, it's just not one set thing, then I don't know where to start. If I think of different rules, then I start and I find out I don't have enough information to do that one. Then I just never know where to go from there.

MK: Right, so you are beginning by a kind of process of elimination. Sometimes having done that, you still don't know?

CI: Yes, I think, 'Oh, it's that', and I get to the end of the formula and I don't know the last variable so I have to think of something else to use. I never know what to do and I waste a lot of time...The teacher always writes in my reports something about being too impulsive, putting pen to paper too quickly.

Caroline's behaviour is, apparently, very typical in situations where students do not know how to proceed with a mathematical problem; according to Schoenfeld (1985, p.42) it is, 'the rule rather than the exception.' He labels as 'naive empiricists' students who rely upon guessing at the nature of a solution, trying it out and, if it works, accepting it without any mathematical justification for its correctness (the examples he gives were of students using compass and set square in geometry to make constructions). Whether it is

appropriate to use this label on Caroline is a moot point since it is clear that she would prefer not to have to rely on guess work but rather to arrive at the solution through a more logical process of analysis and deduction.

Caroline's experimental strategy is mirrored in her actions in the programming class by going to the computer to experiment whenever she has reached an impasse with thinking through the solution steps on paper. Her chemistry example was analogous to the mathematical one (which formula to use to calculate the concentration of a substance?) and also her behaviour was similar.

Was it gaps in her programming knowledge that made it difficult for her to recognize which techniques to use? Caroline did not believe that this was the explanation:

CI: I don't know. Because I always come out with better marks in KU [knowledge and understanding].

MK: So you don't think it's due to a lack of knowledge. It's a certain type of knowledge, how to proceed, that's the missing knowledge?

CI: Yes. I can always remember all the things [that I've been taught], and I know exactly how to do each one...

Finally she returned to her theme of not getting enough practice:

CI: ...that is probably why I can't do problem solving, because you don't get any practice as you go along.

[Interview 25, Caroline, 27/3/95, Transcripts p.365–8]

It seems that the teaching method in programming (and elsewhere in the curriculum) has not provided Caroline with the degree of support that *she* felt she needed to enable her to recognize different classes of problems and the associated techniques to introduce in her solutions. Caroline was unaware of any specific features of the learning resources or of the research intervention that might have helped her in this regard. I discussed with her the design of topics in which the problems presented within a topic were related in some way, for example, through the use of a standard algorithm. This design was intended to build her capacity to recognize the techniques associated with a particular class of problem or situation and to enable her to compare and contrast problems and solution methods. I also discussed with her the kinds of questions that the review process was designed to address, for example:

MK: ...when we were discussing techniques it's not just <u>how</u> they work but <u>why</u> they are used and <u>when</u> to use them. Because if you don't know the answer to the when question then you are stuck, that's what you've told me...

[ibid., p.367–9]

Perhaps the agenda of fostering near transfer within programming (to enable pupils to solve novel problems and to approach such problems with a degree of confidence) was too implicit and rather hidden from pupils.

On the issue of getting enough practice at problem solving, Caroline may have experienced some of the earlier programming problems as too simple and routine to be recognized by her as problems at all i.e. they were not genuinely problematic for her. This would explain her view that she didn't get enough practice at problem solving as she went along, although her reviews of 'posters' and 'address cards' (see 8.2.2) indicate that certain problems did challenge her.

This discussion raises the interesting question of what counts as a problem. Schoenfeld (1985, p.74) discusses the relative nature of the term 'problem' in relation to mathematics, as follows:

> The difficulty with defining the term problem is that problem solving is relative. The same tasks that call for significant efforts from some students may well be routine exercises for others, and answering them may just be a matter of recall... To state things more formally, if one has ready access to a solution schema for a mathematical task, that task is an exercise and not a problem.

Do any useful implications flow from this observation, other than the need for teaching to be differentiated to ensure that each pupil has sufficient opportunities for 'genuine' (in Schoenfeld's terms) problem solving i.e. that there is an element of transfer or connection making involved in obtaining the solution? Being pedantic about the label attaching to a task is unhelpful since a pupil would need to have completed a task and some assessment made of his or her performance in it before the task could be labelled as a problem or an exercise for that pupil. Other than in a one-to-one teaching situation, it is impractical to make such a distinction, and it would be impossible to build any vocabulary or understanding of problem solving processes if one could not be certain that any particular task constituted a problem. Also Greeno (1980, p.12) has raised fundamental objections to this type of distinction being made between a problem and a routine performance:

> The belief that problem solving occurs only when a person lacks critical knowledge about the problem has put us in a position like that of a man who is digging a hole and never gets it deep enough because, no matter how far he digs, he is still standing on the bottom. If a student has learned whatever knowledge is required for solving a problem, then the student can't get credit for problem solving, because the student merely is

> applying the knowledge that (s)he has been taught. Of course, if a student lacks the knowledge needed to solve a problem, the student won't be able to solve it. On this analysis, students are bound to be poor problem solvers, because the only way they can be good problem solvers is by doing things they haven't learned how to do...

Furthermore, Greeno maintains that significant processes such as understanding, planning and organizing activity by setting sub-goals are present in a great many activities that students learn to accomplish routinely, which leads him to conclude:

> These routine activities, therefore, ought to be counted for what they are – namely, as perfectly legitimate acts of problem solving. (ibid., p.13)

Schoenfeld characterizes heuristics as, 'techniques used by problem solvers when they run into difficulty. They are rules of thumb for making sense of, and progress on, difficult problems.' (ibid., p.74) But surely heuristics can also be usefully characterized as techniques used by good problem solvers to *avoid* running into difficulties? When designing a problem solving curriculum one can have as its basis a continuum from simple through to complex problems, recognizing that how a particular problem is situated in the teaching order affects its complexity (e.g. how closely does it mirror previously solved problems?). Heuristics can be introduced through this curriculum and skills in their use built up gradually. When Caroline claims that she, 'can't do problem solving,' she fails to recognize the skills she has developed on the programming course and elsewhere that are amply demonstrated through her performances (see, for example, the discussion of her reviews of 'posters' and 'address cards' in 8.2.2, or the discussion of her work on the topic 14 assessment in 6.3.7). Her dilemma is that of the man trying to dig himself out of the hole. The requirement to synthesize a range of different elements to obtain solutions to later programming problems naturally makes them difficult; they demand higher order thinking.

Paula's experience was quite different from Caroline's; she remained confident about her problem solving abilities throughout, although there were, of course, occasions when she was forced to puzzle (e.g. she talks, during one interview, about, 'trying to figure out,' a solution to a Topic 12 problem and how the idea came to her). Her views on transfer from programming bear this out:

P: Yes [something does transfer]. What we were talking about earlier, the planning of it ,you can have that to do other things, like history investigation ... or whatever.

MK: *Do you think you are more aware now of the need to plan than you were before you began your programming work?*

P: Yes, definitely.

MK: *Do you think that is partly because I have discussed it with you, or has it just arisen through having done the work, or [is it] a combination of both?*

P: Probably both, because I did it in here [in class] and I'm kind of doing it without really thinking about. You said it, I realised, yeah, I do do that.

[Interview 23, Paula, 9/3/95, p.328–9]

Claire was the only person to mention an everyday event as something that she might plan in a top-down way. She mentioned a picnic, 'First you plan where you are going.' Bernadette added, 'You do all the big things first.' Claire rejoined, 'Like the basket!'[57] Of the literature on mathematical problem solving, Nisbet (1990, p.3) comments:

> Too readily it is assumed that the strategies learned will be applicable to problem solving generally. Real-life problems are 'fuzzy': it is often harder to identify what the problem is than to find a solution.

This research has avoided making such assumptions, but nevertheless there must be some 'real-life' situations to which a top-down (or bottom-up) strategy could be applied with success.

Schoenfeld (1985, p.133), as a mathematician taking an outsider perspective of computer programming, observes:

> One can hardly quarrel with the notion of a highly structured, orderly, hierarchical planning process. It may be that such processes are natural in some domains, for example, machine assembly or certain kinds of temporal planning. Successful AI programs in such domains demonstrate that hierarchical planning processes do work. And in messy domains such as computer programming, the idealized representation of the design process as being top-down and hierarchical can serve as a useful goal state for novice programmers. One should keep in mind, however, that idealized behavior is precisely that. In some domains, expert behaviour may not appear to be highly structured at all.

Schoenfeld then discusses competent approaches to problem solving which are 'event driven' or opportunistic, giving as an example, 'While carrying out some minor computations...an individual may observe symmetry in the equations being manipulated. This may suggest that symmetry plays an

---

57 Interview 9, Catherine, Bernadette & Claire, 24/5/94, p.85

important role in the original problem and may call for revision of the entire approach.' (ibid., p.134) One can see that in real-life, solutions are all the more likely to be event driven (Claire and Bernadette might decide to abandon their picnic and go to the cinema instead if it rains!).

Carol-Anne, Bryan and Scott could think of no situations at all, in or out of schools, where they had applied a hierarchical planning process[58]. It seems highly unlikely that there were no such situations (although some of their decision making would be event driven), and much more likely that they had little conscious awareness at the time of their thinking processes, just in the manner that Claudia described – 'you do break problems down, in and out of school, without thinking about it.'

Papert (1980, p.154) observes of procedural thinking: 'Everyone works with procedures in everyday life. Playing a game or giving directions to a lost motorist are exercises in procedural thinking. But in everyday life, procedures are lived and used, they are not necessarily reflected on.' He argues that teachers should tap into this everyday-life experience of procedural thinking so that it becomes better understood and available as a resource for school learning tasks (i.e. his concern here is with transfer in).

The importance of the teacher mediating the process of bringing thinking to a conscious level is evident in Paula's observation (above) if the capacity to transfer is to be enhanced. This does not mean getting learners to think in a very deliberate fashion about every action as they carry it through (which could well impede functioning) rather it is through an on-going process of reflection, prompted by occasions when pupils are required to review their learning.

Furthermore, becoming competent at applying design skills in programming (which does not necessitate an extensive knowledge of language features or programming techniques) may enable pupils to apply these skills more widely in the way that both Paula and Claudia describe, '..doing it without really thinking about it.'

8.5 TESTING AND DEBUGGING STRATEGIES: FURTHER EVIDENCE FROM PHASE 2 & PHASE 3 INTERVIEWS

A range of processes and strategies is encompassed by this heading. In phase 2 interviews (based on the work to the end of topic 10) discussion focused upon

---

58 Interview 10, Carol-Anne, Bryan and Scott, 14/6/94, Transcripts p.104.

three particular aspects; selection of test data; comparison of expected and actual output (related to testing programs); and debugging strategies. In phase 3 interviews (depending upon the stage the pupil had reached) discussion also focused upon methods of tracing a program and more advanced methods of testing programs (based on work in unit 4). The following five questions were addressed during discussions: Do you consider this process or strategy to be necessary? What purpose(s) does it serve? Do you understand the principles of its operation? Can you carry it out effectively? Can you generalize it to non-programming contexts? The first four of these questions I shall consider under one heading, 'The within programming context: purposes, principles and effective application of testing and debugging strategies.' The last question will be linked to one process, comparing actual and expected output.

It would not have been a manageable task to investigate with every pupil each of the several aspects of testing and debugging in relation to each of these questions and sampling across the range of problems tackled; rather discussions took a natural course. Therefore the reporting of this evidence will take a synoptic view and it will draw out interesting issues and comparisons rather than being all encompassing. (There is in any case a full discussion of pupils' solutions to a debugging task – 'prize-draw' – in chapter 5, and further references to testing and debugging within coursework assessments in chapter 6.)

8.5.1 *The 'within programming' context: purposes, principles & effective application of testing and debugging strategies*

In the learning materials, guidance on selecting test data was linked to specific problems, for example, in the 'posters' problem (see 5.3.3) pupils were directed to test the program by varying the details that would be printed on the poster and also the number of posters to be printed out. Comparison of expected and actual output was built into the problem solving process by requiring pupils to predict the output before running the program, so that they would anticipate the responses of the program and check. Alternatively, if the program was generating a number at random then this value would be echoed to the screen to enable the output to be checked.

When questioned about these processes, pupils identified that they were necessary to ensure that the program was correct (further evidence of this is obtained through review task 3). For example, on comparison of expected and actual output:

C: When you work it out in your head and it comes out [on the screen] – then it's matching.

MK: ...why do you think you would do that?

BE: To make sure the programs working.

C: Yes, to make sure.

[Interview 9, Catherine, Bernadette & Claire, 24/5/94, Transcripts p.80–1]

– and making a sensible choice of test data:

S: You have to use the same [actual data] as you work out in your head. It's [to be] simple, simple that you can work out in your head.

[Interview 20, Stephen, 28/2/95 & 3/3/95, Transcripts p.265]

– and systematic testing:

L: Like the first time... you would put in wrong answers [to the quiz questions] so that you would get it all wrong. And then the second one... you would maybe get just some of them wrong, then the rest you would get none wrong.

MK: What are you checking for...?

L: Well you are checking that it is reading the right data [the questions and answers are stored in data statements] and also that at the end it is giving the right statement ['Well done!' etc.] and the correct score.

MK: So there was a message at the end depending on the score?

L: Yes.

MK: Why would you have more than one set of test data?

L: Because if you just have one it might just be like by chance that it was that certain data that it was using that was [making the output] correct.

[Interview 22, Lorraine, 9/3/95, Transcripts p.312]

The guidance on testing was generally adhered to by pupils, according to evidence from the folders of work and assessment portfolios, however this evidence reveals that a few pupils failed to uncover logical errors – testing came to an end when the program ran without generating an error message – or they overlooked certain task requirements, such as prompts for input (see, for example, the discussion of the 'address cards' problem in 5.3.6).

It emerged from pupils' written descriptions of how they identified and corrected the bugs in the 'prize-draw' program in topic 10 (see 5.3.8) that a variety of strategies had been used, mostly with success. To recap, in addition to testing the program with different inputs, responding to error messages, and using the help hints (according to questionnaire responses), some pupils

worked from the printed listing to spot particular error types and/or referred to a previous design or program.

When I asked Carol-Anne and Scott to explain to me how they would locate any bugs in a program, Scott indicated that he always began by checking the printed listing:

MK: *...why would you start with the printed listing Scott?*

SC: Just to see, see if you can see anything if it sticks out it's wrong, it's clearly....

CA: There's too much on the screen, you can't see – if it's a long program it's all away back you've only got a wee bit on the screen.

MK: *...What sort of things would you look for when you were looking at your printed listing, what are the obvious errors that you think you would be trying to spot...?*

CA: Variables.

MK: *What do you mean by variables?*

CA: The numeric ones and the string ones.

MK: *You'd check that you've used the right type of variable – what else?*

CA: Forgetting comment marks whenever you put the things in – you just write it in and you get 'SYNTAX error'.

MK: *Is that the quotes you mean?*

CA: No the two dashes [// – signifies a remark in the program].

MK: *...Anything else that you would look for?*

[Interview 10, Carol-Anne, Bryan and Scott, 14/6/94, Transcripts p.105–6]

Papert (1980, p.113) observes of debugging: 'It always takes time to trap and eliminate bugs. It always takes time to learn necessary component skills. What can be eliminated are wasteful and inefficient methods.' Carol-Anne's and Scott's strategy is obviously designed with efficiency in mind. They seem to have appreciated that guessing at the nature of the solution and trying it out at the computer (the 'naive empiricist' approach) is not the sole or most efficient route to debugging programs.

According to the evidence on planning processes, many pupils found it helpful to produce a top-down design because it reduced logical errors in the program, and also having a design to work from made it easier to locate bugs in a program during testing at the computer (see 8.4.1) or to make subsequent amendments to it. Thus planning strategies and testing and debugging strategies are 'partners' – they operate together to enable pupils to produce

correct solutions efficiently. Pupils who are not good at planning out solutions may be able to compensate by developing really good testing and debugging strategies (Chris falls into this category), and vice versa.

Also, according to a number of pupils, writing out the program prior to entering it could further reduce errors, therefore leading to greater efficiency, for example:

> ***Claudia:*** *...I suppose it's necessary [to write out long programs] since otherwise I'd make a lot of mistakes. You notice mistakes on paper that you might not notice on the screen.*
>
> ***Claire:*** *...you can see what you're doing before you put it up [on the screen].*
>
> ***Bernadette:*** *Because if you were just typing it into the computer, you would have to change it hundreds of times, if it didn't work. But if you write it down you just have to copy it onto the computer.*
>
> *[Interview 8, Claudia, Kirsty and Kelly, 10/5//94, Transcripts p.65; Interview 9, Catherine, Bernadette and Claire, 24/5/94, Transcripts p.74]*

One of the potential dangers of presenting the problem solving process as a systematic progress through a range of stages – analysis, design, implementation and evaluation – within each of which there are further stages, and of rehearsing pupils' problem solving skills in this way, is that pupils may not come to view the process holistically. Evidence to the contrary is provided by pupils' ability to appraise the contributions of designing a program and/or producing a written version of it towards arriving at a correct solution efficiently. The specific focus upon each stage of the solution process may actually have assisted pupils to do this (e.g. the 'posters' problem is split into six tasks) in comparison to a less structured approach.

Methods of tracing an algorithm or program were discussed at length. The methods that were introduced to pupils consisted of: (i) a hand trace (or dry run) – producing a written step-by-step account of the workings of an algorithm or program; (ii) a trace table – showing the order in which statements are obeyed and how the values of any variables alter as the program is obeyed; and (iii) inserting an extra statement into a program to enable the programmer to trace a variable by echoing its value to the screen.

Was the concept of tracing an algorithm or program understood? Not readily, according to Caroline: '...I read that [the part about hand tracing] over and over and I still couldn't understand...'.[59] Gerard's explanation of a hand trace was

---

59 Interview 25, Caroline, 20/3/95 & 27/3/95, Transcripts p.344

very tentative, 'Copying a program? I don't know.'[60] Carol-Anne gave a simple but accurate description of its purpose:

CA: I think that you had to check that the program was doing everything you wanted it to do.

[Interview 24, Carol-Anne, 10/3/95, Transcripts p.340]

Paula was confident to explain:

P: Going through all the program like a line at a time, just looking at what it does.

MK: Does that mean that you just look through it once in the order of the commands?

P: No. It's going through it in the way that the computer would do it.

[Interview 14, Paula, Chris, Kirsty, 20/12/94, Transcripts p.172]

Following in some cases a brief revision or explanation of the technique, four different situations were identified where doing a hand trace of a program could be helpful: it could be used to check the logic of a program before it is entered at the computer (4 pupils); it could be used to help you to understand the operation of a new program (5 pupils); it could be used to locate a bug when an error has been revealed during testing at the computer (7 pupils); or it could be used to assist you to construct a new program by thinking through its operation at a detailed level (1 pupil). This last situation struck me as similar to the process of constructing a flow-chart. Just over half of these responses were volunteered without any prompting or hints.

However there is no evidence on whether pupils used these techniques, other than when called upon to do so. Perhaps pupils went through the process of tracing an algorithm or program in their heads without writing anything down? Carol-Anne's and Paula's descriptions of a hand trace make it seem the natural thing to do if your program isn't working and you want to understand what's wrong with it, or you want to get to understand a new algorithm or program. Also doing a trace in your head has certain advantages (as well as obvious disadvantages) over setting it out more formally on paper; it takes less time to do, you do not need anything to hand other than the design or listing; and it is a private activity so that no one will discover that you are having any problems with understanding. However pupils need to be able to perform a trace accurately otherwise it will produce misleading information, and they may therefore need help to use it.

---

60 Interview 13, Gerard, 2/12/94 & 9/12/94, Transcripts p.153.

Perkins and Martin (1986, p.227) use the term 'close tracking' to describe the procedure of thinking through the actions of a program. It is prompted by such questions as, 'what will what I have written really do?', or, 'how did my program get that wrong answer?'. They view these two questions as having special significance:

> ...close tracking is the critical filter that allows detecting programming errors with understanding. To be sure, running the program to see what happens also acts as a critical filter: If the program fails, there is something wrong. However the critical filter of running the program provides far less information than the critical filter of accurate close tracking. While the former simply presents the programmer with the fact of an error, and perhaps an error message or some anomalous output, the latter leads the student through the program's action blow by blow.

A further mechanism used within this study to encourage self-prompting was the use of 'what if...?' scenarios to cue pupils towards close tracking an algorithm or program. Getting pupils to predict the outcome of making certain (specified) changes to a program and then testing the prediction at the computer was a common question format.

Pupils' references to spotting errors on the listing or comparing listings do not necessarily signal an attempt to understand, at a detailed level, the workings of a program. However much of the evidence points towards the high value that pupils placed upon understanding, and it is apparent from performances and discussions that pupils did frequently come to understand their errors. For example, one of the most persistent problems for Scott, Bryan and Carol-Anne in topics 9 and 10 had been putting quotes around the variable name or using a constant value when trying to output the value of a variable. These problems were teased out during discussion:

MK: *...you had 'PRINT' and quotes and the variable name inside the quotes – do you know why that doesn't work?*

SC: It would just come up on the screen.

MK: *Literally it's the variable name that comes up on the screen. Whereas what you really want it to do is to output....*

CA: Any name

MK *.... the value of the variable, whatever the user has entered, Carol-Anne.*

CA: I done that in the poster, I think.

MK: *... Bryan you had 'PRINT', quotes you had 'Celtic versus Rangers'. Why would that not work?*

B: Because you want it to be different, [not] just print that all the time.

[Interview 10, Carol-Anne, Bryan & Scott, 14/6/94, p.96]

Schoenfeld (1985, p.73–4) identifies three ways in which the complexity of heuristic strategies and the amount of knowledge necessary to implement them have been under-estimated, each of which is relevant to programming. Firstly, a heuristic is often used as a label for a category of closely related strategies ('tracing' provides a computing example), therefore it needs to be unpacked. Secondly, implementing a strategy may be more complex than at first appears since there may be many separate phases of implementation each of which is a potential cause of difficulty, therefore teaching needs to be precise and rigorous. Thirdly, the successful implementation of a heuristic strategy often depends heavily on a firm foundation of domain-specific resources.

Building pupils' knowledge of testing and debugging strategies is clearly quite a complex process. This is because it isn't just a matter of teaching one strategy or 'rule of thumb' that can be straightforwardly applied across the board. There are many subtleties and choices to be made about how best to proceed in a given circumstance. This has been recognized and accommodated through a method of teaching which relies upon modelling of solution approaches and careful exposition in order to introduce pupils to a range of strategies in a natural way. This may explain why most pupils have experienced success with testing and debugging and are confident about their skills.

8.5.2 Generalizing beyond programming

When prompted to consider the strategy of comparing actual and expected output applied to non-programming contexts, pupils gave an interesting range of examples which encompassed: doing an estimate before using a calculator (suggested by Claire and Bernadette, but also recognized by others as a common situation); doing the inverse calculation to check an initial result (Stephen); working out the unknown of an equation by inspection before simplifying it (Dawn); having derived the solution substitute values back into an original formula to check that the solution is correct (Stephen and Paula); testing a hypothesis in science (Stephen); calculating in advance the expected output from a spreadsheet calculation (Stephen) and testing a spreadsheet by varying the inputs systematically (Paula). Stephen provided particularly well differentiated examples (the first example below is of a spreadsheet calculation and the others are based on maths).

S: You've got rainfall or something. Add them all up in case you've made a mistake with the formulas [on the spreadsheet] and it's maybe multiplying or something.

MK: What was the formula to do, can you remember?

S: To add each box on under each other.

MK: Were you totalling the rainfall?

S: Yes.

MK: Did you work out in advance what you ought to get as the total [on the spreadsheet]?

S: Yes.

MK: Did you just do that yourself, or were you prompted to do it?

S: Just done it out of habit because we do it in maths or whatever, just count them up yourself.

MK: What about situations where you are using calculators in maths, what do you tend to do there?

S: If it's something simple I just do it in my head because there is no point taking the time to use the calculator.

MK: Okay. If you have got a complicated calculation to do and you would then choose to use the calculator, do you do any checking in your head?

S: It depends if it's something with the science function on the calculator, there is not much you can do to check it, I don't think.

MK: Say it was just an ordinary calculation, multiply two numbers, but quite difficult numbers?

S: Just get your answer and divide it by the other one, see if it gives you the other one. Just use the calculator again.

MK: Oh, so you actually just do the process in reverse?

S: Multiplication, and then divide it.

MK: You do the inverse of it to check that out, okay. Were you ever taught to do an estimate as a means of checking that your calculations are correct on the calculator?...

[Interview 20, Stephen, 28/2/95 &3/3/95, Transcripts p.265–7]

All of the strategies suggested by pupils (or discussed by pupils in response to my suggestions) share one or two common purposes – by anticipating the correct solution, the problem solver is enabled to *monitor* that the solution is on course and to *check* the accuracy of the final solution. However the above strategies do not all operate in identical ways – some are more closely analogous to the comparison of expected and actual output in programming than others – and they are applied across a range of situations. It is their common purposes that bind them, and therefore recognizing what a strategy is

designed to do is a necessary precursor to seeing its broader applicability. The clear implication is that teaching needs to focus on why a strategy is needed or useful, as well as on how and where it may be applied, in order to foster transfer through a process of mindful abstraction.

Stephen's examples, and also his discussion of the 'TRIS' model of planning and writing English essays and how this compared with planning processes in programming (see 8.4.5), demonstrate his awareness that a strategy does not always operate in an identical way when translated to a new context; the operation may be similar in some respects but different in others. This awareness is very important, for two reasons mainly: he will be able to see connections even when there are some surface differences, and it will prevent him from applying a strategy more widely in inappropriate ways (negative transfer). According to Perkins and Salomon (1989), teaching on strategies, if it is to be effective, must stress the need for a strategy to be moulded to the new context of application.

8.6 CONTRASTING ALTERNATIVE SOLUTIONS AND EVIDENCE ON NEAR TRANSFER

Generating and contrasting alternative solutions to a problem engaged pupils in creative and critical thinking. Such activities also provided a means to extend pupils' knowledge and understanding of computing concepts, principles and techniques and their metacognitive skills.

The generation and comparison of alternative *programmed* solutions was a feature of certain topics. The comparisons arising from one such topic (topic 13) are discussed below in 8.6.1.

As part of the Standard Grade course all pupils solve a range of problems using an integrated software package, such as Microsoft Works, learning to use and integrate the different components of the package – word processor, spreadsheet, graphics, database – and to recognize which tool(s) to select for a particular problem. Thus there is an opportunity for pupils to make comparisons between two solutions where only one is programmed and for seeking evidence on near transfer from programming. The relevant findings are discussed below in 8.6.2 in relation to pupils' spreadsheet solutions.

```
     Survey on rainfall - September
    ==================================

You have to enter the amount of
rainfall for each day in millimetres.

The program will calculate and output
the monthly rainfall.

Day 1
Please enter the rainfall in mm > 4
Day 2
Please enter the rainfall in mm > 1
Day 3
Please enter the rainfall in mm > _
```

Figure 42: First version of 'rainfall statistics' program – input screen format (unit 3 topic 13).

8.6.1 *Contrasting alternative programmed solutions*

The problem on rainfall statistics, first introduced in chapter 6 when discussing coursework assessment tasks and evidence on attainment (see 6.3.2 and 6.3.6), provided fertile territory for exploration.

Two versions of the solution are developed in the topic. In the first, the user enters data to the program (figure 42), whereas in the second the data are inserted by the programmer and read automatically from data statements when the program is run. One of the reasons this problem proved so interesting is that it led to a discussion on the design of the user interface at a stage when pupils were just beginning to explore ideas on this. The first discussion, which took place with Kathleen and Claudia, arose from their observations on testing the first version of the program, which they found to be time-consuming and error-prone. (Since there is no error trapping in the program they would have to begin again whenever a wrong value was entered.)

KA: I didn't like having to like put in all the numbers all the time. See like every time you wanted to do the program.

Cl: [Agrees with Kathleen and adds], like if you made a mistake ....

In spite of these drawbacks, it was still viewed as more suitable for the user than the second version:

KA: Because it's [the first version is] easier and they [the users] get instructions so that they know what to do.

MK: Where are the instructions?

KA: When they appear on the screen.

and:

KA: Because it's [the second version is] too difficult, they [the users] would need to know what line to put in and what commands and things.

So how could the first version be improved to overcome the problem of having to re-enter all the data if an error is made?

Cl: [Thinks] Some way to be able to maybe delay one without having to go back to the start. If you ....

KA: . ... If it says something like.... maybe if you press return again or something.

Might the program be able to detect a wrong value automatically?

KA: It would only be if it knew like the numbers.

MK: *[Hint] What if the user typed in 3,000 for the rainfall statistic?*

KA: No, because [you can] give it a maximum or a minimum.

[Interview 17, Claudia and Kathleen, 27/1/95, Transcripts p.209–11]

I framed the discussion by introducing two perspectives, that of the user and the programmer. There was one occasion when I made a suggestion that the computer could detect an error (which was not responded to immediately) and another occasion when I hinted about a mechanism that would allow the computer to detect some types of errors (values that were out of range). These measures enabled Claudia and Kathleen to come up with suggestions on how to improve the first version of the program.

The discussions with Caroline, Paula and Lorraine were similarly conducted.[61] As with Claudia and Kathleen's interview, prompts and hints were successfully employed. The following interview excerpts provide some further examples:

- Suggestions on remedying the problem of having to start again to re-enter all the data if an error has been made at the input stage:

Caroline:

CI: Well as you input the value it could ask you to do it twice or something so that you check it... You have entered this number and then you have to enter it again, or something like that.

Paula:

P: It could give them a choice after they have entered everything at the end, 'Do you want to alter this?', and if they say yes then it could go up [to the start] and then they just keep going down [the list] until they get the one [that needs correcting]... Or you could ask them, you know, 'What one is it that you want to change?'...

- Could you get the program to detect automatically a wrong value being entered?

Lorraine

L: No I don't think so, because it couldn't rain one day and the next day it could rain a lot and the computer wouldn't be able to tell.

MK: *[Hint] I wonder, if, say, you entered a negative value?*

L: Yes, it would be able to then because it can't be below zero because there is either ....

---

61 Interview 25, Caroline, 20/3/95 & 27/3/95, Transcripts p.351-4; Interview 23, Paula, 9/3/95, Transcripts p.325-8; Interview 22, Lorraine, 9/3/95, p.304-7.

MK: *Right, so if you build into your program some sort of check that indicated that it has to be at least zero. [Prompt] Anything else that you could build in as a check?*

L: Well it couldn't be like over a certain number, because if it was too high then it wouldn't be logical ....

- How would the program respond to an invalid entry?

Lorraine

MK: *Let's say the user enters negative five, what would happen next? What would you imagine would happen on the screen?*

L: It would probably have one screen to say like, 'It's not possible', or a, 'Bad value'.

MK: *Then what?*

L: It would probably prompt you to re-enter the number.

MK: *Say you then entered negative eleven?*

L: Then it would do the same until ....

MK: *When does it stop doing that?*

L: When you do one above zero.

MK: *Or...?*

L: Or beneath the highest number.

- How can you build this feature (a range check) into the program?

Caroline

CI: If it is greater than 20 then print 'this is wrong' or something.

MK: *So put an error message on the screen. And then you would obviously have to let the user type in the new value again... Would it do it just once or would it keep on doing it until the user got it right?*

CI: Yes it would have to keep going, it would have to be a loop so that it would keep going until they got the right answer.

MK: *So what kind of loop would be best?*

CI: I don't know.

MK: *Would it be repeating a fixed number of times or is it until ....*

CI: Yes, until.

MK: *What's the condition?*

CI: Until rainfall equals .... between 0 and 20.

Thus contrasting the two solution methods led to an open-ended discussion that teased out the different perspectives of the user and programmer, which is something that a number of pupils found difficult to do (see 7.2.8), and it laid the foundations for the work in later topics on the design of the user interface and methods of validating input. A discussion on this theme is clearly generalizable to any situation where a computing solution is being developed.

8.6.2 *Comparisons between spreadsheets and programming, and evidence on near transfer*

The two aspects discussed below are pupils' comparisons between practical problem solving using COMAL and practical problem solving using a spreadsheet, and near transfer. Did pupils experience any transfer from programming? Beyond this, are there any indicators on whether near transfer has occurred and by which mechanisms (low road and/or high road)?

It is necessary first to sketch the background – how pupils were taught about spreadsheets, the types of problems that they were set to do, and their performances in certain tasks.

8.6.2.1 The introductory spreadsheet topic and 'league tables' project

The introductory spreadsheet topic contained two closely related problems. The first problem description set out the following requirements. The spreadsheet had to present the 1993 monthly rainfall statistics for two Scottish weather stations. Four statistics – total annual rainfall, and average, minimum and maximum monthly rainfall – had to be calculated from the monthly statistics for each station. In addition, the output had to indicate whether each month was above or below average for rainfall and the monthly statistics for each weather station had to be displayed graphically. The second problem involved tailoring the first solution to process statistics on hours of sunshine, and this item could be included in the coursework assessment portfolio. This problem was presented in two parts, as a general level (G) core (up to the point of calculating the four statistics) and a credit level (C) extension, in order to map it onto the syllabus levels. A detailed marking scheme was produced.

Pupils had to arrive at the solution to both problems independently. Since the second problem was very similar to the first (only the data, labels and two of the formulae had to be changed), it provided, in effect, an opportunity for pupils to 'polish up' and document the solution for inclusion in the coursework folder. Pupils were required, as in programming, to develop the

solution systematically by analysing the problem, making a plan (of resources and solution steps), implementing their plan and evaluating the solution.

Before embarking on the first problem pupils had spent three periods examining simpler problems (a 'tuck shop' account, for example) in order to learn the basics of spreadsheet operation and vocabulary. (Although current 5–14 curriculum guidelines include learning to use a spreadsheet as part of the mathematics curriculum, pupils in this class had never used a spreadsheet before.) They also learned to use more up-to-date computers which supported a 'windows' environment (the class was held in an adjacent lab). The dual task of becoming familiar with the operation of the computer and the software[62] in a short time presented a real test of pupils' computing expertise.

There was no formal teaching directed towards any aspects of the problem or the solution method which all pupils should have encountered previously in their programming work or elsewhere. These aspects included concepts such as data, input – process – output, numeric value, text, formula, arithmetic expression, test data, printed output, formatting of numbers and text, adjusting layout, and processes such as planning a screen layout, loading software, entering, editing, saving, printing, testing, debugging and evaluating. Statistical and mathematical concepts – sums, averages, maximum and minimum values, charts etc. – would be familiar already.

Pupils worked from an outline which set out the two problems and the main solution stages. They made reference to school produced documentation to learn how to carry out processes such as editing or replicating formulae. This mode of presentation was intended to capitalize on the knowledge and skills that pupils had gained through learning to program and to further develop their skills in independent learning. There was some limited assistance to enable pupils to see connections between programming and spreadsheet constructs (e.g. 'You will have used this kind of notation in programming: IF difference>0... ').

Beyond the introductory spreadsheet topic pupils could opt to tackle a credit level spreadsheet project, *league tables*. The class teacher had selected this project from amongst a batch of projects issued by the SEB, all of which were accompanied by detailed marking instructions. (The introduction to the project which the pupils would read is shown in figure 43 overleaf.)

---

62 Pupils used Acorn Archimedes computers running Advance – an integrated software package.

League Tables Spreadsheet Project

Every week teams from sports clubs all over the country play against each other in competitive leagues. The results of games are used to calculate points for each team: 2 points for a win, 1 point for a draw and 0 points for losing. A new league table is published each week with the teams sorted into order by points.

If teams have the same number of points they are sorted further by goal difference. Goal difference is calculated by subtracting the number of 'goals against' from the number of 'goals for'. Goal difference is not normally included in the final, published version of league tables, but is required at the processing stage.

Often the new tables are needed very quickly. For example on Saturday evenings the newspapers and television stations must have the tables ready almost as soon as the results come in. So a computer-based system is appropriate.

Figure 43: Introduction to the *league tables* spreadsheet project (SEB, 1993a).

	Sunshine statistics coursework:		**League tables project (C)**
	(core – G)	**(extension – C)**	
No. of pupils attempting task	19	10	17
No. of pupils completing task	17	8	16
Ave. % mark for pupils completing task	86.2%	73.9%	87.2%

Table 20: Summary statistics on performances in spreadsheet tasks.

Pupils had to create a week 1 and week 2 league table for eight teams, devising their own test data, and produce user-documentation. Various totals had to be included in the spreadsheet to check on the accuracy of the data, e.g. that the total of 'games drawn' is an even number, and the 'games won' and 'games lost', 'goals for' and 'goals against' totals are equal. A full report had to be completed, to include a detailed evaluation of the solution against a range of criteria and a diary of progress.

8.6.2.2 Performances

Table 20 (opposite) presents summary statistics relating to pupils' performances in the coursework and project tasks. Marking procedures were the same as for the programming coursework evidence (appendix 4 shows three examples of how I applied the marking criteria to pupils' work on the spreadsheet coursework item).

Michael chose not to attempt the coursework task at all and Stephen and David did not complete the core task. Fewer pupils attempted or completed the extension task. Stephen, Anthony and Gerard chose not to attempt the *league tables* project, and Michael did not complete it. Other items of non-programming coursework or a different project could provide evidence of attainment in the practical abilities element, and hence the reason for there not being a complete set of evidence. Pupils who completed the core but did not attempt the extension coursework task performed almost as well in the core as the others – there was less than 2 marks (7.8%) of a difference, on average – which suggests that lack of time or interest in the task, and/or the availability of other evidence to contribute to the summative grade were the main factors that influenced choice.

For the *league tables* project, the average number of entries in pupils' progress diaries is nine periods (almost eight hours), which is much less than the 14–20 hours indicated in the syllabus to complete a credit level project (SEB, 1991, p.22). Some of the performances in the project were outstanding (Paula gained 100%). It would seem to have been quite an easy project for this class to do, when one considers how little time in class that pupils spent on it and the high average mark in comparison to the average mark for the extension coursework task. Perhaps the class teacher and I were expecting too much of pupils in the coursework task. The way in which the marking scheme for the project is constructed allows pupils' performances in analysis and design, implementation and evaluation to be separately gauged. The strongest

Grades	Spreadsheet Coursework	Spreadsheet Project	Programming Coursework
Bryan	3	2	4
Stephen	7		4
Claire	3	2	3
Scott	2	1	4
Paula	1	1	1
Claudia	3	1	2
Carol-Anne	3	1	3
Dawn	1	1	4
Chris	1	1	1
Caroline	3	1	1
Anthony	5		4
Michael		7	4
Kirsty	2	1	1
Bernadette	2	1	3
Kelly	3	1	2
Catherine	2	1	2
Kathleen	1	1	2
David	7	1	2
Gerard	3		4
Lorraine	3	1	1
Key:			
1, 2 – Credit grade		3, 4 – General grade	
5, 6 – Foundation grade		7 – Fail grade (task not completed).	

Table 21: Grades for spreadsheet coursework, spreadsheet project and programming coursework.

Grades	Spreadsheet Coursework	Spreadsheet Project	Interview(s)	Reference in transcripts
Chris	1	1	14	p.167–8
Kirsty	2	1	14	p.167–8
Paula	1	1	14, 23	p.167–8, p.319–22
Kelly	3	1	15	p.187–9
Catherine	2	1	15	p.187–9
Michael		7	19	p.243–5
Lorraine	3	1	22	p.314–5
Carol-Anne	3	1	24	p.334–6
Caroline	3	1	25	p.362–4
Key:				
1, 2 – Credit grade		3, 4 – General grade		
5, 6 – Foundation grade		7 – Fail grade (task not completed).		

Table 22: Interviews in which pupils compared learning to program with learning to use a spreadsheet.

performance overall was in analysis and design (93.8% on average), then in implementation (85.8%) and evaluation (81.2%).

The grades awarded to individual pupils are shown opposite in table 21 (only the overall grade is shown for the coursework assessment). The programming coursework grade is also included for comparison. Four pupils – Bryan, Scott, Dawn and Bernadette – did much better in the spreadsheet tasks. The other pupils' spreadsheet and programming grades were fairly close (discounting the grade 7's which indicate 'task not completed'). Reaching topic 13 in programming was not a predictor of performance in the spreadsheet coursework task. This may be because the weather statistics context was a familiar and straightforward one and did not need any introduction, and all pupils had accrued sufficient programming experience for this not to be a factor.

8.6.2.3 Pupils' comparisons between programming and using a spreadsheet

During seven of the phase 3 interviews, nine pupils (table 22) compared their experiences of programming with learning to use a spreadsheet. The timing of these interviews coincided with when pupils were doing or had recently completed spreadsheet work.

A range of issues was discussed centring around the following questions.

> Which (spreadsheet or programming language) did you find easier to learn? Why do you think that was?
>
> Which (spreadsheet or programming language) gave/would give the best results for this particular problem?
>
> Which other problems (from unit 3) could/could not have been solved easily using a spreadsheet?
>
> Were you aware of any programming techniques or ideas or experiences that helped you with your spreadsheet solutions?
>
> Was it worthwhile to learn both (spreadsheet and programming language)?

I shall examine pupils' responses to the questions, taking, as before, a synoptic view of the evidence to highlight a range of responses. It should be borne in mind when interpreting the evidence that pupils' experiences of building spreadsheet solutions was rather limited – much more so than for

programming – and this was especially true for pupils who were interviewed earliest, since work on spreadsheets was on-going.

- Which (spreadsheet or programming language) did you find easier to learn? Why do you think that was?

It was easier (some thought much easier) and quicker to learn to use a spreadsheet.

Michael

M: I've just started to put my formulas in and I've to bring in my sheet with the table that I wrote out on it [showing the week 1 data].

MK: That's good then. Did you find it easier to learn to use the spreadsheet than to learn programming?

M: It was easier with the spreadsheet, yes.

MK: A lot easier?

M: No not really, it never had as much explaining, but it's easier to do anyway.

Paula

P: ...with the spreadsheets you were only really entering the information as a formula and that [the spreadsheet] does all the work for you.

MK: ... presumably before you know what to enter you have still got some thinking to do, Paula?

P: Yes, but not as much, I don't think.

- Which (spreadsheet or programming language) would give/gave the best results for this particular problem?

Opinions varied on this question. For example, in relation to the rainfall statistics problem, Catherine and Kelly thought that it made no difference. When asked to make a general comparison, Michael highlighted the greater variety of output that can be obtained using a programming language:

MK: Do you think ...the output is better with a spreadsheet ...?

M: No, well with the programming you get a lot of different stuff outputted though. The spreadsheets are just spreadsheets.

MK: It's just tables and numbers, you mean?

M: Yes.

MK: Have you tried any charts – you know, sort of bar graphs or anything like that?

M: No I've not got that far yet.

However Carol-Anne preferred the spreadsheet output because it was better set out and clearer:

CA: It was more like a table, the spreadsheet, and it was clearer to read it.

- Which other problems (from unit 3) could/could not have been solved easily using a spreadsheet?

The intention behind this question was to draw out pupils' knowledge of the different characteristics of a programming language and a spreadsheet. Pupils identified that there were certain problems in unit 3 that could easily be solved using a spreadsheet (such as the problem on currency conversion) and/or others that could not (such as the problem on producing a variety of patterns on the screen, from Topic 12). Michael was given a prompt (think of the problems we've talked about earlier in the interview) and Lorraine and Caroline a suggestion (think of this problem) to overcome the difficulty of having to come up with examples 'on the spot'. Being able to repeat and vary an action and interact with the user were two of the features identified by pupils as being not easily reproduced using a spreadsheet (or any of the other software tools they had learned).

However Caroline considered that her most useful solutions were those obtained using the other software tools:

CI: Well they [your programs] do teach you things about the way the computer works and that, but probably I would say the other ones were more useful.

Paula weighing up the advantages and drawbacks of each as follows:

P: ... just what you were saying there, if you are using the individual one [say a spreadsheet or database], they are better and you can get better results, but it's only that one thing that you can do ... With programming you can do spreadsheets and databases and everything all in the one.

- Were you aware of any programming techniques or ideas or experiences that helped you with your spreadsheet solutions?

Responses varied on this question. Catherine and Kelly saw no connections between programming and using a spreadsheet (apart, one presumes, from the very obvious one that both involve the use of a computer to solve problems), and Caroline could not recollect anything in particular. However further prompting soon revealed to Caroline a number of connections, such as the use of branching:

MK: ... you were able to say which months were above and below average in sunshine – how did that work?

CI: It was, if it was greater than the average then that meant it was sunny, and if it was lower than average, it was dull, and you used the formula.

MK: Does that link to any programming ideas that you've used?

CI: Yes, well in the 'If' and 'Then' things we did [in programming], if this is greater than whatever then. .., so it was the same idea.

I asked Caroline if she had been aware of these connections at the time, when she was working on the problem:

CI: No not really, probably because I'd already learned it then and I didn't notice.

MK: .... you would use the ideas without thinking about them?

CI: Yes, just used it.

Paula gave an example to illustrate the connections that she saw, which refers to planning, checking and testing processes:

P: ... like for the football [league table spreadsheet], we wrote out what it was meant to look like, like the totals at the bottom to make sure that it worked and everything was right, and then when you entered it, [you checked] if it was different. You planned out the steps that you were going to do, like you were going to make it out on paper then enter it then print it. So that you knew everything that you had to do and you didn't miss anything out.

- Was it worthwhile to learn both?

Pupils appreciated that considerable time and effort had gone into learning to program and therefore this was the aspect that had to be weighed up most carefully. The benefits of learning to program were seen as: for working life; understanding better how a computer works; confidence (being able to use computers properly and fix problems); being better at planning.

The benefits of learning to use both tools were seen as the variety of activities on the course and the wide range of problems encountered. Caroline was however very critical of the time spent on programming when she realized that there were to be no programming questions in the final written exam:

CI: ... it is the other topics that you don't have enough time to spend on that come up in the exam because you spend all the time doing programming.

It is recommended in the syllabus that 25% of the course is given over to programming however the programming coursework grade contributes only about 13% to the overall award. To restore the balance, pupils would need to do a programming project so that the project grade is also drawn from programming, but this means that they get less practice on topics which *are*

assessed in the external exam. Having to spend longer than the recommended time on programming adds to the time pressures.

8.6.2.4 Taking the evidence together on pupils' performances and self-appraisals

The number of pupils opting out of the extension part of the coursework assessment suggests that some pupils felt under pressure of time or were not confident or keen to continue with the task. Ideally, I should have asked all pupils about their reasons for not continuing, but the period leading up towards the standard grade examination is a difficult time to arrange interviews when pupils are trying to focus ahead. For the same reason, the discussions on comparing the two tools (programming and spreadsheet) could not be conducted with everyone, but had to be fitted into some of the phase 3 interviews if time permitted. The constraints of situating the case study within an examinable syllabus, and also of resources and time, prevented a full exploration of transfer effects, which might have encompassed far as well as near transfer.

Programming was the main practical problem solving activity that pupils engaged in prior to embarking on spreadsheet work. The evidence on pupils' performances in the spreadsheet coursework and project tasks suggests positive findings on (near) transfer for the majority of pupils. It is also a fairly robust demonstration of transfer – not entirely spontaneous, but not cued either by any instruction designed to systematically help the pupil to contextualize methods in the new domain.

Consider the situation if transfer from programming had not occurred. Then most of the aspects of learning to use a spreadsheet as a tool for solving problems would have had to be taught afresh. None of the additional time spent on programming would be recouped, which could have an adverse effect on pupils' performances in other aspects of the course (Caroline's concern). The class teacher had been confident that the extra time spent on programming could be afforded and would benefit pupils, not only by enhancing their programming performance but also by enhancing their problem solving performances across the board. Her confidence seems to have been borne out by the evidence.

By which mechanisms might transfer have occurred? Because Catherine and Kelly saw no connections between programming and using a spreadsheet, it is tempting to conclude that if transfer did occur from programming, it was via

the low road for them (and perhaps also for most other pupils). However this is unlikely to be the case entirely, for the following reasons:

(a) Even in terms of very basic IT competences (loading, saving, viewing work, editing etc.) which would seem to be the most likely candidates for transfer via the low road, these operations are carried out differently on the two computer systems.

(b) The solution mechanisms also do not have much surface resemblance. For example, to total or average numbers using COMAL an algorithm has to be devised giving the precise solution steps to be followed by the computer. To do this on a spreadsheet requires that the data is entered into a row or column, and a formula entered into a different cell to sum or average the values across a given range of cells.

(c) Because there is such a wide range of aspects that could potentially transfer from programming – Salomon and Perkins (1987) identify six broad categories ranging from mathematical and geometrical concepts and principles to cognitive styles and enthusiasms and tolerances – these would need to be carefully unpacked and mapped onto the transfer domain, in the manner that Caroline and I began to do when discussing, before pupils could readily identify which aspects did or did not transfer. To expect pupils to be able to do this for themselves without guidance or the opportunity for extended reflection, is quite unrealistic.

(d) To solve any non-trivial problem using a computer generally demands some thinking and effort.

Many of the concepts taught through programming and frequently encountered in that setting are general computing concepts, such as input – process – output, types of data (numeric, text) etc., and these would be recognized by most (all) pupils in a spreadsheet setting. These aspects would most likely have transferred by the low road – pupils would simply have put them to work in the new setting without much or any deliberation about them.

In terms of problem management strategies – adopting a *consistent, planned and systematic approach to obtaining a computer solution to a problem* – this aspect could transfer from programming by either mechanism. The conditions for low road transfer – varied and frequent practice to near automaticity – could certainly be met for most pupils given the consistent emphasis on

Review Task 3

From your personal experience of learning to program, what advice would you give to someone who was learning to program for the first time?

Your advice should be designed to help the person to write *good programs* and to go about it in an *efficient manner.*

Try to list at least ten different points. Give a reason or reasons for including each of your points.

Here are two examples which you may choose to include in your list.

Read the problem description more than once, and write it out in your own words to make sure that you understand the problem properly before you start to solve it.

Organize your printed listings, sample runs etc., as you go along and don't let them pile up. This will make it easier to refer back to your previous solutions and will save you time later.

Write your list of points on the sheet attached to this one. You can produce a joint list, or separate lists each if you want to list different points. Put your name at the top of the sheet.

Figure 44: Review task 3.

Groups	
Dawn, Bernadette, Gerard	Paula, Chris, Kirsty
Catherine, Kelly	Scott, Bryan
Claudia, Kathleen	Anthony, Stephen, Carol-Anne
Individuals	
David	Caroline

Figure 45: Review task 3 – pupil groupings.

adopting a planned and systematic approach to obtaining programmed solutions, when combined with the long duration of the study. The conditions for high road transfer could also be met for those pupils who have been enabled to reflect on and articulate problem solving strategies, and to bridge to non-programming contexts. The dual possibility of transfer mechanisms operating together strengthens the study considerably.

When analysing the potential for transfer and pupils' performances (did they demonstrate high road or low road transfer?), I found it helpful to think in terms of a continuum (rather than a dichotomy) with the 'middle road' sketched in. The middle road – where pupils' actions are neither mostly the product of deliberate, mindful abstractions nor of reflexes – seems a better fit for the day-to-day learning activities that pupils engaged in. Stated generally:

- Pupils frequently worked on problems that had elements in common with previously solved problems (the next problem may have been an extension of the previous problem) – we wished pupils to see the connections clearly or to seek out connections actively;
- At the same time we wished pupils to exercise a degree of caution in case some common elements were invoked differently in the two situations (thus guarding against negative transfer effects);
- We also wished pupils to be aware of the novel aspects of problems, otherwise motivation may decrease as the work took on a sameness.

Caroline's review of two related problems (see figure 39 in 8.2.2) encompasses all three of these elements, as did some interview discussions with other pupils (although in some instances the third aspect evaluated negatively).

8.7 EVIDENCE FROM REVIEW TASK 3

8.7.1 Outline of review task 3

Pupils were briefed on the third – and final – review task (figure 44) at the end of each phase 3 interview. This group task was intended to provide a meaningful context for pupils to reflect on their learning during the programming course and to share insights with others. Pupils formed into pairs or threes (figure 45) and discussed for about fifteen-twenty minutes, without the class teacher or I being present during the discussion. Catherine and Kelly met in Kelly's home one evening to discuss. David and Caroline opted to do the task individually as a written exercise since their respective

partners (Claire and Lorraine) were absent. Unfortunately there was no further opportunity for Claire or Lorraine to do the task since the class was moving on to revise other topics. Michael opted out.

During the briefing I stressed that it was important for pupils to *use their own ideas* – the two points within the task outline were only there as examples and they need not be included. The issue of what would constitute a good program and an efficient manner was left for pupils to unpack.

Why frame the task in this way? The task could, instead, have asked pupils to, 'list the features of a good program' or to outline this or that programming technique (as one might do in an examination question), however I wasn't interested in textbook responses, rather I wished to tap into pupils' personal, and hopefully broader, understandings. The task could also have asked pupils to appraise their programming abilities by posing the question directly, e.g. 'are you good at solving programming problems?', however I thought it unlikely that most pupils would justify their responses fully, and therefore I would be unable to glean much from what they had written, and also modesty might prevent an honest appraisal. Therefore the question was posed in such a way as to tap into pupils' actual experiences. By specifying an audience that wasn't just the class teacher or myself, I hoped that pupils would base their advice on approaches that had genuinely worked for them, rather than merely parroting any advice that they had been given.

8.7.2 Findings from review task 3

The findings are presented in two ways; firstly by summating across the responses, making links to earlier themes; and secondly by examining the responses of individuals, making links to earlier discussions.

8.7.2.1 Themes

A very wide range of themes emerged, from very practical matters (such as protecting one's password carefully) to philosophical points (about mental approach). Appendix 5a lists each theme, includes some quotations under each heading, and indicates the number of responses in each category.

The most common themes were: on saving and printing work; on keeping your work organized; on making sure that you understand (the problem, your program, any new techniques); on what to do if you are unsure or get stuck; on retention and revision; and on collaboration with other pupils. There is no

evidence anywhere of 'surface approaches' to learning being advocated or of silly approaches being suggested.

Pupils communicated their ideas very clearly and succinctly. It is difficult to see how this set of advice, if appropriately communicated, could fail to help any third year pupil starting out on programming. The general approach of thinking ahead to anticipate potential problems and taking appropriate actions to avoid them, rather than merely reacting to situations, comes through strongly. Pupils pass on their personal knowledge of what these potential problems are and how they would deal with them.

It is unfortunate that this class could not have met with a third year class to pass the advice on directly – they would have made very credible and personable witnesses – but the way in which the timetable is constructed makes this impossible. With minor modifications, the advice could be extended to any learning contexts where pupils are given some responsibility to decide on how they will manage their learning, but would be of limited use in a situation where the teacher calls the tune at every turn.

The set of advice is best characterized, not as, 'tips for learning programming', or even as, 'tips for studying', but rather as 'guidance on how to manage learning effectively in the programming class'. It does not focus very much on any detailed technical aspects of programming – not because pupils would be incapable of providing such advice, but rather because the learning materials serve this purpose well and, in any case, such guidance would not be easily summarized onto one sheet of paper! It is richly contextualized, unlike the typical advice on study skills where learning is envisaged to take place in isolation (e.g. 'Have you got a quiet, familiar and uncluttered environment in which to study?', from a current Jordanhill Library pamphlet). It goes far beyond revising, ignores exam tips, and is ideally pitched to younger learners. Taken together, it makes one very aware of the complexity of the task that pupils faced in order to manage their learning well, and of how much they have learned in the process.

The advice is sometimes provisional – there is no assumption that one method will work best for everyone: 'If you feel that you can work with a partner successfully it can be very helpful.' It takes account of task variables: 'Always read new techniques thoroughly, ask for help if you need to because they will turn up again,' and people variables: 'Don't change your password too often or you will forget what you have changed it to. Mrs xxx [the teacher – not me!] will go mad ...'. It even includes some aspects that the class teacher and I

did not pay much attention to, but which are clearly important: 'Take your time to learn where the keys are on the keyboard. This makes your keying in faster for long programs.'

It attempts to prepare the learner for the difficulty of the work by advising on strategy e.g. 'Go through it one step at a time, don't try to do the whole thing at once,' and on other aspects such as expectations: 'Don't expect it to be easy, it's not all games'; perseverance: 'New things may seem hard, but they become clear after a few tries'; and being prepared to accept help: 'Be prepared to accept help from other pupils ...'.

How did pupils construe writing 'good programs' in an 'efficient manner'? A *good program* would appear to be one which solves the given problem (and not some erroneous interpretation of it), which is the product of a design process (including systematic testing) and which is readable and therefore maintainable. An *efficient manner* encompasses such aspects as: learning the pre-requisites for programming (e.g. how to use the keyboard), monitoring your understanding, having good retention, engaging in planning, following your design through to implementation, utilizing the full range of available resources (people, the learning materials, the computer to obtain feedback), being well organized and avoiding practical pitfalls. Efficiency has not been equated with fast progress, but rather with *careful and planned* progress. These are aspects which could transfer widely, even beyond formal learning contexts.

8.7.2.2 The responses of individuals

Appendix 5b sets out the responses of groups and individuals to the third review task. It is disappointing that Michael opted out of the review task. Michael was one of five pupils who had been dissatisfied with his progress on completing the second unit (see 7.2.11) and this may explain his reluctance to provide advice for others (I did not press him for his reasons). How did the other four – Bernadette, Gerard, David and Bryan – and also Stephen, who was furthest behind and did not complete unit 2 – respond to the review task? Many facets of the issues that they had discussed with me resurfaced in a positive light.

Bernadette had mostly been concerned about her understanding which she felt had suffered because she rushed to keep up with others in the class. Much of her advice (and Gerard's) is geared towards avoiding this problem:

Take your time, because if you rush you will not remember how to do a certain thing e.g. how to loop may be in more than one problem. If you don't know how to do it first time, it will make it more difficult the second time.

Get the teacher to explain anything you're not sure of.

Discuss with your friend.

Keep your diary up to date so you can remember what you done the last day, and can ask the teacher if you had difficulty with any of the previous work.

Get your work corrected so you know if you've made any mistakes and you can then go back and correct any errors.

If you have been off or missed any work, come up at lunchtimes, as it can be easy to fall back on the work.

Also Stephen's advice homes in accurately on a number of the aspects which clearly hampered his progress (including not always taking his work seriously):

Write in your dairy at the end of every period to save time and effort finding out where you are in the book.

Keep your folder tidy and keep all printouts in order so you don't get in a muddle.

Save everything every so often so that you don't lose any work.

Try to memorize all the new techniques you learn so that you don't have to keep backtracking.

Don't expect it to be easy, it's not all games.

David has included some advice on designing programs and keeping up with the class (a matter which concerned him greatly):

Try to follow your top level designs very closely.

Although the Mode 7 design grids are at first pointless, hang on to them as they are helpful as your program nears completion.

If you feel you are falling behind then take your workbook home and do as much as you can.

Bryan (working with Scott) has come up with a commendable problem solving schema, which includes analysis, planning, and testing and retesting of the solution, but, unfortunately, no review of the solution:

- *Read the task carefully, because it helps to know what you are going to do.*
- *Plan the steps in your head so that you have a clear idea of what's going on. This will also save time.*
- *Make sure your plan of action works, write your ideas down.*
- *Go on the computer and start your task.*
- *Run your task to see if it works.*
- *If it doesn't get Help Hints to help you, if that fails then you need a teacher to lend a hand.*
- *Once it's fixed run it to see if it works.*

- *Get a printed listing and a sample run.*
- *Arrange the printouts to save time.*
- *Save the program!*
- *Operation Complete!*

This systematic schema, if applied, would be helpful to avoid some of the haphazard errors in Bryan's solutions.

Of course, some of the pupils' advice could turn out to be short lived – since pupils may lack the application, ability or opportunity to follow their own advice consistently. But this possibility does not detract from the value of the exercise since it is designed to elicit pupils' self-knowledge, which is a first step towards self-regulation of learning and thinking (Nisbet, 1980).

8.7.3 Discussion of the findings from review task 3

The review task has very successfully demonstrated some of the 'learning about learning' that has taken place during the programming course, and the importance of promoting autonomous learning to provide the opportunities for such learning to take place. It also demonstrates how negative experiences of learning can be turned around into positive experiences, to enable future learning tasks to be tackled, hopefully, with greater confidence and skill.

A common criticism of the conventional advice on study skills is that it fails to transfer; it is presented in isolation from the context in which students are actually learning. There is also doubt about whether some of it is good advice, since it doesn't appear to fit well with the actual way in which successful learners go about their learning:

> The most telling criticism of this conventional advice is that many of the most successful students do not adopt the recommended procedures.
> (Nisbet and Shucksmith, 1986, p.15)

These criticisms cannot be levelled against the pupils' advice, since it is based upon their recent classroom experiences and practices (or sometimes failed practices turned around and stated positively).

Nisbet and Shucksmith list other counts on which the study skills movement can be criticized (for completeness I shall list all six counts):

1. It lacks a theoretical basis, having no link with developments in cognitive psychology.
2. It lacks an empirical basis, being based on a self-perpetuating consensus.

3. It is often too general and out of context, so that it is not seen by learners as relevant to their needs and so is not applied.
4. It is not transferable, being merely a collection of tips for coping with specific subject-based procedures.
5. It can too readily become merely a way of coping with the formal requirements of the school system, particularly with passing examinations.
6. It is too late, in that habits are already formed by age sixteen or eighteen. (ibid., p.26)

What can pupils learn from sharing learning experiences and insights with each other? Much more, it would seem, than they might learn on a conventional study skills course. On each of these counts, the pupils' advice stands up quite well. There is a theoretical basis (pupils take a constructivist perspective on learning); it has an empirical basis; it is neither too general nor out of context; it does not consist of just a collection of tips for coping with specific subject-based procedures; it focuses on deep approaches to learning rather than merely on passing exams; and there is a better prospect that habits can be reformed since the advice stems from self-assessments.

8.8 SUMMARY AND DISCUSSION

This section extends the earlier discussions in this chapter by drawing together four related strands: pupils' strategic knowledge, their ability to generalize strategies from programming, transfer from programming, and how each of these interfaces with the teaching approach.

8.8.1 Developing pupils' strategic knowledge and their ability to generalize strategies from programming

The evidence vindicates the teaching method in which specific language features, process skills, standard algorithms, programming concepts and principles, and problem solving processes are thoroughly integrated in the teaching, and each aspect was subject to review and reflection during the course.

Linn's (1985) 'chain of cognitive accomplishments' is not reflected in the findings. This theoretical model places the learning of language features as the first link in the chain, the learning of program design skills (to encompass templates and procedural skills – mainly planning, testing and reformulating a plan if any tests fail) as the second link, and the learning of problem solving skills (generalized design skills) as the third link. The findings from the review tasks and related interview findings reveal that pupils can discuss

meaningfully and in broad-ranging terms a range of program design and problem solving skills *at any stage of learning programming*, given the opportunity to learn them.

Linn's model is very helpful to unpack the components of programming instruction that may lead towards the development of problem solving skills, but it is not helpful as a teaching order, and it is clearly not always accurate in terms of how pupils' knowledge actually builds, since this obviously reflects how they have been taught. This argument is supported by Resnick's (1987, p.48) analysis:

> Thinking skills tend to be driven out of the curriculum by ever growing demands for teaching larger and larger bodies of knowledge. The idea that knowledge must be acquired first and that its application to reasoning and problem solving can be delayed is a persistent one in educational thinking. "Hierarchies" of educational objectives, although intended to promote attention to higher order skills, paradoxically feed this belief by suggesting that knowledge acquisition is a first stage in a sequence of educational goals.

Through the perpetual learning of prerequisites, Papert (1980) argues, students can lose all motivation, and powerful ideas and the intellectual aesthetics of subjects can also be lost. Instead intuitively well understood contexts should be provided to enable learners to reason and to problem solve. He draws an analogy to a good art class, in which the child learns technical knowledge as a *means* to get to a creative and personally defined end which both the teacher and pupil can be genuinely excited about.

Just as with every other aspect of pupils' programming work, there were variations amongst pupils in their ability to generalize strategies from programming by suggesting other contexts of application. However the relationship between pupils' ability to generalize strategies and their ability to apply them in programming was not a straightforward one. For example, Stephen's explanation of the top-down design process in programming was rather hesitant and, by his own account, he found some aspects of it difficult to apply (specifically, knowing which steps are main steps and working out the refinements). However he identified a situation in his English class which involved a hierarchical planning process and could explain it well to me. So long as a strategy is understood in principle, even if it is sometimes difficult to apply in programming, there would seem to be some prospect that pupils can generalize it. Salomon and Perkins (1987, p.162) observe that mastery of programming is not strictly necessary for high road transfer:

> strategies and principles might transfer by abstraction and application elsewhere even when they are not fluent yet or when the knowledge base specific to programming is not sufficiently consolidated so that learners could use them to full effect in programming.

The significance of this is that every programming student, and not only the most advanced or talented, might benefit from the teacher mediating the process of connection making. Therefore Linn's (1985) decision to investigate the third (problem solving) link of the 'chain of cognitive accomplishments' with only the most talented programming students – the top 1% of 2,400 middle school students – may have missed out on some vital evidence that would have cast doubt on the theory.

It is probably not a coincidence that writing an essay in English was discussed by a number of pupils since it is apparent that the teaching on this particular writing strategy was very explicit. Indeed, since pupils spend much longer in school learning writing skills than they do learning programming, it would make sense to focus on the potential for 'transfer in' to programming as well as 'transfer out'. However one has the impression from the interview discussions (e.g. with Caroline) that not much teaching on strategies within the formal curriculum was explicit.

8.8.2 Transfer of problem solving strategies from programming

The evidence on pupils' performances on spreadsheet solutions indicates positive findings on near transfer from programming for the majority of pupils. Although being able to apply a strategy successfully in programming is not a prerequisite for being able to generalize it, it may affect the *likelihood* that it will transfer. Paula made repeated references to the benefits of sub-dividing complex problems in programming, and it is therefore likely that she would actively seek to apply the same 'divide and conquer' strategy elsewhere, feeling confident that she would reap the same benefits from its use. Indeed she refers to having used this approach across a range of circumstances – when obtaining spreadsheet solutions, doing an essay in English, and for her history investigation. On the other hand, although David understood the principle behind top-down design and could identify other applications, laterally he found it difficult to apply in programming and was not confident about using it. Since he had experienced little leverage from having used it, either in programming or English, it seems unlikely that he would actively seek out further opportunities to apply it. Therefore it is important that pupils learn to

use strategies skilfully in programming, not only because it enhances their programming abilities and confidence, but also because it will provide the impetus for their wider application.

Of course, there are factors beyond this which affect pupils' ability to transfer strategies from programming, particularly if far transfer is sought. A particular barrier to far transfer by the low road is that, 'the formal context of programming does not look or feel very much like the tense context of a labor dispute or the excited context of hunting for a new stereo system.' (Perkins and Salomon, p.23) This factor may explain the paucity of pupils' suggestions on wider applications to everyday thinking.

Also practice and feedback are equally as important in the transfer domain as in the original domain of learning, but how can these be delivered within the confines of subject teaching in the secondary school, unless teachers work collaboratively in inter-disciplinary teams to ensure a coherent approach? Without this, one could envisage that a computing teacher's well meaning suggestion that a particular strategy could usefully be applied and practised in, let's say, physics – might be treated as unwelcome interference by the physics teacher, and the suggestion could indeed be wrong (this is never or rarely a useful strategy in physics), or inappropriate to the physics curriculum or the pupils' stage of learning. Also it would take a confident child to deviate from the normal way of doing things in the physics class.

The infusion method of content instruction advocated by Swartz and Parks (1994) involves getting students to engage thoughtfully with what they are doing and also to focus on the particular thinking skill they are learning. If this approach was adopted in every subject where skilful problem solving plays a part, then there would be much more reinforcement of problem solving skills. But this is a big 'if' – the approach has to be fitted to the existing curricula across a range of subjects, and the move could be strongly resisted by secondary teachers if they perceived that there would be any dilution of subject content because of the greater emphasis on thinking skills.

Local knowledge will also affect pupils' ability to transfer strategies successfully. To take a programming example, when decomposing a problem an efficient strategy is to re-use existing templates whenever possible, but a pupil lacking in knowledge of available templates cannot apply this strategy successfully. The same argument applies in any other learning situation.

8.8.3 *Other potentials for transfer from programming*

There are other potentials for transfer beyond problem solving strategies demonstrated in the findings from this chapter and earlier findings. Some can be broadly encompassed under the heading 'approach to learning'. They include:

- planfulness (thinking ahead, considering how best to proceed, anticipating problems and preventing them from occurring);
- adopting a systematic approach to a problem and systematic working habits (saving regularly, organizing materials etc.);
- self-monitoring (e.g. of understanding, progress) and self-regulation of learning;
- collaboration, and independent learning.

Willingness to persevere and to expend effort have been linked in previous discussions to pupils' conceptions of learning and intelligence – whether entity or incremental. From the advice that pupils chose to pass on to younger pupils about how to cope with the difficulty of the work and on learning for understanding, the need for perseverance and effort would appear to be important aspects of pupils' learning on the course, which does suggest that there has been an influence on pupils' self-conceptions.

8.9 CONCLUSIONS

The main conclusions arising from pupils' reflections on learning and problem solving are indicated below:

1. Experiences from the first review task indicate that it is important not to rely on written reviews as the *only* means of getting pupils to reflect on their learning. Such an approach will not be effective if it becomes just another classroom routine and a demand on pupils' time. Teachers must share with pupils their personal knowledge of the benefits of reviewing learning regularly, and provide a range of opportunities for review.

2. The majority of pupils could discuss, in a detailed and wide-ranging manner, program design skills (to encompass testing and debugging) at an early or interim stage of the course. Some pupils could also generalize these skills by identifying a range of non-computing contexts where they could be applied. Towards the end of the course the majority of pupils

applied these skills successfully to obtain spreadsheet solutions, indicating that near transfer has likely occurred. This evidence validates the teaching method which seeks to develop pupils' strategic knowledge throughout the course.

3. Pupils had very different experiences of learning programming and their difficulties were located in different areas which took some time to tease out during interviews. Discussions focused on many facets of learning, encompassing knowledge and strategies, near and far transfer, the pedagogy associated with the course, self-awareness (metacognition), and confidence. Many of the difficulties resurfaced in a positive light when pupils considered the advice they would offer to novice programmers, and I have no doubt that the interview discussions played a part in this.

4. Enabling pupils to make valid connections between two programmed solutions or between programming and other activities depends upon developing their critical and creative thinking skills and ensuring their understanding of problems and solutions. There is no short cut to this, as the evidence presented in this, and other chapters, reveals. It is a painstaking process that requires a well designed programme of activities, interaction between pupils, effective teacher mediation and time.

5. The findings, particularly from the third review task, demonstrate clearly that 'learning to learn' has occurred, and that the learning environment has enabled pupils to become very self-directed. There is no recommendation from pupils of surface approaches designed to bring about fast progress, rather efficiency is equated with careful and planned progress on a problem, and with understanding and retention. Pupils' advice on strategies indicates their awareness of the need to be systematic, to self-monitor and self-regulate. However not all of the advice is strategic – there are implicit and explicit references to perseverance and attitudes.

6. The timescale of the intervention, spread out over one-and-a-half school sessions, has been a very significant factor in its success. This has provided the opportunity for a range of contexts for reviewing learning, for pupils to assimilate to the teaching approach, for pupils to develop a deep understanding of the subject matter, for effective collaborations to be established, and for pupils to become self-directed. It has provided ample opportunities to examine their progress in learning across a wide range of dimensions, and to establish relationships of trust between all participants which have greatly enhanced the quality of the data obtained. A more

concentrated schedule (the work could have been fitted into twenty school weeks) would not have permitted these things to happen to the same extent.

7. There would seem to be many potentials for transfer from programming to other formal learning contexts, according to these findings. These require further investigation in favourable circumstances. To assess this potential more accurately, it would have been very interesting, for example, to witness the English department's teaching on writing critical essays and pupils' problem solving performances across a range of subjects.

9 CONCLUSIONS

This chapter discusses the main conclusions on the bearing of the instructional and research methods on the quality of the educational outcomes. The conclusions from chapters 5–8 are integrated into the discussion and any implications are drawn out.

9.1 UNDERSTANDING AND COMPETENCE IN PROGRAMMING

Pupils did learn to program. All pupils performed well in the two items of evidence used to derive the programming coursework grade. The marking criteria were detailed and stringently applied. Their programs were well designed, ran without generating any error messages, contained few logical errors, and were correctly sequenced, modular, user friendly and readable. This lays a secure foundation for any more advanced programming (at Higher Grade or university) that pupils may go on to do.

The four stage framework of analysis, design, implementation and evaluation enabled the majority of pupils to make systematic progress on problems. Within each stage strategic thinking played an important role in enabling pupils to solve problems with reasonable efficiency. During interviews pupils discussed a wide range of strategies for designing, testing and debugging programs. Their programming behaviours exhibited far less of the undesirable characteristics noted of beginner programmers in 'typical' classrooms, such as failing to plan or reflect, rushing straight to computers, constant floundering, and using random trial and error to debug programs.

Through these performances pupils have demonstrated a good understanding of programming and problem solving. Some programming concepts were difficult for many pupils to understand when first introduced (such as the distinction between numeric and string variables). Further opportunities for contextualized practice and feedback (using techniques of scaffolding and fading) and review – such as is illustrated by topics 9 and 10 – were effective to strengthen pupils' understanding.

Some powerful design and debugging strategies, such as top-down design and close tracking, were difficult for many pupils to apply correctly, although most pupils could explain the underlying principles of how each strategy operated and what it was designed to do, and appreciated why it was useful in programming. These are complex strategies to learn and apply. Their successful application is always highly dependent upon domain knowledge, for example,

one cannot close-track a program accurately when lacking precise knowledge of the effects of each language construct contained in the program. Clear articulation of the processes, wide experience of applying them to programming situations, occasional coaching, and reflecting on their use afterwards provided effective means to build pupils' competence and confidence in applying them.

The notion that complex ideas can take time to bed in is therefore an extremely important one to communicate to beginner programmers who are struggling to understand and to learn. It is an important notion to communicate to some teachers who are too eager to intervene by providing the solution or substituting a simpler problem. It is also an important notion to communicate to designers of programming (and other) syllabuses who fail to appreciate the complexity of the ideas contained in their curricula and who attempt to pack too much content in. The evidence points clearly towards the majority of pupils having adopted a deep approach to learning, in spite of the pressures which many experienced to cover ground. Discussions with pupils indicate that they monitored their understanding carefully and took a range of steps to safeguard it.

The nature of the evidence retained in the assessment portfolio enabled a wide range of programming competences to be validly assessed, however the process of assessing the class's performances across a range of different tasks (including the formulation of detailed assessment criteria specific to each task) was painstaking and time-consuming. The restrictions imposed by the SEB summative assessment requirements make it desirable to position summative assessment opportunities at the end of the course and to use the assessment portfolio as a purely formative measure. This could result in there being fewer summative tasks which would make the assessment more manageable.

A potential danger that could have arisen from the choice of programming as a vehicle for developing generalizable skills is that, if the majority of pupils had found the experience of doing programming to be too perplexing, this would create unfavourable conditions for such skills to develop. Many studies on programming indicate that students generally find programming difficult to learn – despite Clancy and Linn's claim (see 2.7.1) that using a case study method makes it easier! However the emphasis on learning for understanding and self-pacing, the social support built into the learning environment, and other aspects of the teaching approach prevented this situation from occurring. Learning programming was still not easy for pupils, but it proved to be a

manageable proposition, and, as a consequence, its wider potentials could be tapped within this study.

9.2 METACOGNITION AND HIGHER ORDER THINKING SKILLS

Being meticulous about checking work in progress and any finished work and making the necessary adjustments and corrections was a factor which led to some pupils performing better than others. However the majority of pupils did pick up on errors (e.g. nearly all design errors were corrected) and achieved successful outcomes without relying on the teacher to check their work. This provides evidence on the metacognitive skills of monitoring, checking and self-testing. There is also ample evidence on planning, much of it imposed by the programming regime – read the problem description carefully, perhaps discuss it, plan the screen layouts, design the program, write out the program – before going to the computer. However planning also occurred spontaneously, as can be seen, for example, through the deliberate way in which certain pupils used the revision questions and summary sheets to serve specific learning purposes – carefully reading the summary sheets and then doing the revision questions to consolidate learning, delaying doing the revision questions to assess longer term retention (which would not have been permitted by any teacher operating a strict homework regime), and so on. Planning also occurred when pupils chose their method of attack for debugging – taking a printout of the program and scrutinizing it, checking against the design, checking against an earlier program, and a long list of other possibilities revealed by their descriptions. Metacognition clearly lies at the heart of successful problem solving, which indicates the wisdom of encouraging pupils to self-prompt in the manner that is frequently highlighted in this thesis: not to pose to themselves very complicated questions that might result in failed attempts, but rather to pose simple and natural questions at every stage of the solution process.

In response to the final review task, pupils made no recommendations on surface approaches designed to bring about fast progress, rather efficient progress in learning was equated with careful and planned progress on a problem, and with understanding and retention. Pupils' comments stressed the importance of various aspects of self-management and self-regulation and how to approach a complex learning task. The case study environment has therefore enabled pupils to 'learn how to learn' by strengthening their metacognitive skills whilst also supporting pupils to take responsibility for their learning.

Higher order thinking skills came into play through the requirements to: analyse problems; design, implement and evaluate solutions; document, modify and enhance existing solutions; generate and contrast alternative solutions; synthesize elements of previous solutions; learn and apply new techniques; and review their learning. All pupils were able to demonstrate higher order thinking across a range of performances – practical, written, and spoken. There were naturally many occasions when pupils failed to apply a thinking skill (such as analysis or synthesis) with success – these occasions are well documented and discussed in the findings. The intention of this study was not to turn pupils into superb thinkers overnight, but rather to foster their thinking skills carefully and to encourage an incremental conception of learning. I believe that the case study has been successful in this.

However pupils' evaluation skills may have benefited from more systematic attention, since emergent problems during design and implementation tended to swamp their solution attempts, thus overshadowing later aspects of the solution process. Pupils could have been asked to evaluate a given solution – it could be another pupil's solution – against the problem description and a set of criteria either provided by the teacher or, better still, devised by pupils. In a similar vein, pupils could have been asked to analyse a given problem without designing or implementing a solution, in order to learn how to deal with problems that are not well defined. This would involve having to identify any hidden aspects of the problem, resolving ambiguities, and setting boundaries on the problem. The main emphasis on the course would continue to be upon the integrated use of strategies to solve problems in their entirety, but an occasional spotlight would be placed on a specific thinking skill or process to extend pupils' competence in applying it. Thus the two contrasting infusion approaches discussed in chapter 2 (the first exemplified through the ideas of Polya and Schoenfeld, the second through the ideas of Swartz and Parks) can be brought closer together.

Being able to draw valid comparisons between solutions and solution methods depends upon developing critical awareness and a sound understanding of particular problems and solutions. These may be fostered through a carefully constructed programme of activities, effective interactions between pupils, effective teacher mediation and unhurried opportunities for pupils to think. The constraints of time – twenty pupils is a large number for the class teacher to get round in less than an hour, especially when technical hitches arise – and pressure for coverage combined to make it difficult on occasions for all of these

things to happen. Since fast progress is never guaranteed on a problem it can be hard to justify the time spent on programming in comparison to other more routine activities which can be seen to make a more direct contribution to performance in the final written examination, especially during S4 when pupils are focused ahead on examinations. Therefore the class teacher has reduced the course content a little to enable pupils to complete their programming work in S3. She has also introduced some new measures to encourage pupils to monitor their progress from the outset of the course which allows earlier intervention to prevent some pupils from falling behind.

9.3 GENERALIZING STRATEGIES FROM PROGRAMMING AND NEAR TRANSFER

The interview discussions revealed how some pupils could generalize problem solving strategies from programming to other (school) problem solving contexts. These discussions were very testing for pupils since problem solving is poorly articulated across the curriculum and strategies are often taught narrowly (as tricks of the trade). Such discussions were designed to enable pupils to reach a better understanding of why, when and how certain strategies come into play, to raise awareness of problem solving across the curriculum, and to enhance the prospects of transfer occurring through a deliberate process of abstraction and connection making (high road transfer) in either a forward or backward reaching direction.

Because secondary pupils have a daily exposure to teaching across a range of subjects they may actually be in a stronger position than their teachers to assess the generalizable aspects of courses. The detrimental effects of subject compartmentalization are however very difficult to overcome. How do secondary teachers get together to share their knowledge of the curriculum when so much of their time is class committed? And how much collaboration is there between the working parties set up to devise national syllabuses in different subjects? None, in my experience of being a member of such a group (although the examination officer in our group also had responsibility for physics, he made no inputs to draw our attention to any commonalities between computing studies and physics).

I am confident that near transfer has occurred from programming (to practical problem solving using a spreadsheet) for the majority of the class. This conclusion is not reached through comparing the performances of a control and treatment group or two different treatment groups in pre- and post-test measures, which is the common method to reach such a conclusion. Rather it

is based upon a three stage process of: (1) ascertaining the potentials for transfer by identifying the common elements in the two situations; (2) a careful examination of the teaching processes leading towards pupils' performances in the spreadsheet tasks; and (3) a detailed assessment of individual performances (which is illustrated in an appendix but not discussed at length). However the constraints of the situation prevented a fuller investigation of transfer effects, which is called for if any firmer or broader conclusions are to be reached. According to the responses to the final review task, there would seem to be many potentials for transfer from programming to other formal learning contexts which would require further investigation in favourable circumstances.

9.4 AFFECTIVE AND SOCIAL OUTCOMES

The questionnaires encompassed a wide range of indicators on pupils' affective responses to learning to program. They were effective as a tool to support pupils to review their learning in an on-going manner. The main problem that I identified with linking this review process to the completion of a unit was that some pupils took a very long time to complete a unit, and therefore their reviews were too infrequent and spread across a number of terms. The two measures that the class teacher has since introduced with programming classes (mentioned above) should reduce this problem considerably.

Pupils' self-assessments (of understanding, confidence etc.) were carefully formulated and were generally quite accurately reflected in their performances. Therefore using questionnaires supplemented with interview discussions can provide an effective means to gauge and further develop pupils' metacognitive skills. The information so derived can point teachers towards appropriate actions to strengthen pupils' learning.

The majority of pupils expressed positive views on the learning resources and the way in which the programming class was run. In this and any previous evaluations conducted within the programming project, almost all pupils have indicated a preference for self-paced learning. The whole ethos of the case study environment was designed to promote independent thinking and to enable pupils to be self-directed. Pupils seemed to benefit from the added flexibility of classroom transactions in comparison to more traditional 'teacher led' approaches. They operated flexibly in their interactions with others, maintaining a pace of learning that suited them individually whilst still being able to discuss problems, compare solutions, combine resources, or get help

with problems. A supportive atmosphere developed amongst pupils as the course progressed. The relationship between the class teacher and the class was, I thought, exceptionally good. Her interest in pupils as individuals and the genuine pleasure that she expressed in their achievements – either to me or directly to them – clearly contributed to this.

Not all aspects of the course evaluated positively. Almost half the class were uncertain or unhappy about their progress by the end of the second unit, and some were disappointed because they did not get far enough with the course to do a credit level coursework task (this is in spite of having done extra work at lunchtimes and after school). It is now widely acknowledged by computing studies teachers that the programming topic is too content-packed. Statistics disclosed to me by the SEB on take-up of credit level projects speak volumes on this issue – 61 pupils opted for or were allocated by their teachers to do one of three programming projects, and 602 opted for or were allocated to do one of two applications projects (from a sample of 2151 candidates in the 1995 examination). The message to teachers is obvious – if you want your pupils to get good grades, then allocate an applications project.

Almost half of the criticisms made by pupils in response to the question, 'Were there any things that you didn't like about the work?' were about laboriousness. Some pupils just did not like writing, but it is also clear that the COMAL programming environment contributed to the problem. A move to a newer programming environment is called for, one which provides better scaffolding for the processes of designing, entering and debugging programs. Pupils' ability to understand the hidden processes of program execution could also be enhanced through the choice of a programming environment which makes these processes more visible and manipulable.

The girls' progress and programming coursework grades were better, on average, than the boys'. The girls' questionnaire and interview responses reveal that they valued any opportunities to consolidate and review their learning. This was less obvious from the boys' responses (they would speak of repetition).

Two of the cases discussed at length were Stephen's and Bryan's. The discussion with Stephen (see 7.2.11) revealed a range of factors that compounded to slow his progress. Better self-management would have enabled him to overcome or avoid some of the problems. Did he lack the motivation or the capacity to manage his learning better? More individual

teaching could have helped to reduce his frustration. Bryan's performance was erratic at times (see 7.2.6). He claimed that he made errors because he was bored, but having to spend time fixing errors would add to his boredom. Other of his comments indicate that he found the problems to be dull and not fun (he mentioned topic 13, on processing weather statistics).

A degree of general disaffection with school may have depressed the efforts and progress of certain boys, perhaps even substantially (Michael's sudden dip in progress was one of the first findings to be presented). This theory would need further investigation. It would also be worthwhile to target interventions on certain pupils (such as Stephen) whose method of working is disorganized, to enable them to manage their learning more successfully.

9.5 ENCOURAGING PUPILS TO REFLECT ON LEARNING AND PROBLEM SOLVING

Mixed findings from the reviews of particular problems that pupils had selected to write about indicate the importance of not relying on this or similar approaches (daily dairies etc.) as the *only* means to get pupils to reflect on their learning. Such approaches will not be effective if they become just another classroom routine and a demand on pupils' time. Teachers must share with pupils their personal knowledge of the benefits of reviewing learning regularly, and provide a range of opportunities for review. Some pupils lack the skills to structure their ideas in writing and therefore cannot communicate effectively through this route. The stimulus of listening to other pupils' ideas and suggestions is also absent, which makes it harder to think of something to say.

Responses to the second review task and interview discussions afterwards on problem solving strategies learned in the first unit, demonstrate in a convincing manner the benefits to be gained from direct teaching and reflection on problem solving strategies taking place from the outset of the course, rather than this being delayed until after the 'basics' of programming (i.e. language constructs) have been learned. Had there been a better foundation to build on from pupils' problem solving experiences in other school subjects and from primary and S1/2 stages, then more insights on generalizing strategies from programming could have been anticipated.

Pupils had very different experiences of learning programming and their difficulties were located in different areas which took some time to tease out during interviews. Many of the difficulties resurfaced in a positive light when

pupils considered the advice they would offer to novice programmers (in the final review task), and I have no doubt that the interview discussions played a part in this.

The extended timescale of the research intervention has provided opportunities for a range of contexts to review learning, for pupils to assimilate to the teaching approach, for pupils to develop a deep understanding of the subject matter, for effective collaborations to be established, and for pupils to become self-directed. It has provided ample opportunities to examine their progress in learning across a wide range of dimensions, and for relationships of trust to be established between all participants which have enhanced the quality of the research outcomes.

9.6 SOME FINAL REFLECTIONS ON THE RESEARCH

The research demonstrates how an appropriate balance can be struck between content objectives and broader educational goals within an infusion approach. Ideas on learning with understanding and teaching for understanding have been successfully combined with other ideas on the thinking curriculum and learning to learn, resulting in balanced educational outcomes having been achieved.

I believe that this study has provided useful pointers on fruitful areas of investigation on transfer from programming and fruitful methods of investigation. However the ideas need to be taken much further to investigate how pupils are actually learning about problem solving and thinking skills across the curriculum, to bring the approaches closer together across subjects (and this to be reflected in national advice and curricula), and to look for the effects of an intervention in one subject area on aspects of performance in another, taking account of pupils' metacognitive insights.

This was not a particularly easy topic to research into. It demanded of me the same qualities that I sought to develop in pupils. The challenge of bringing some of the ideas contained in the research to the student teachers whom I have daily contact with keeps me thinking deeply about the issues. I would wish to take this work forward in the kind of directions that I have indicated above – addressing the problems which are 'subject specific' (investigating, for example, better programming environments), trying different ways of getting children to reflect on their learning, and also taking a more cross-curricular focus on the issues.

LIST OF REFERENCES

Baron, J.B. (1987) Evaluating Thinking Skills in the Classroom in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Biggs, J. (1988) The Role of Metacognition in Enhancing Learning *Australian Journal of Education* Vol. 32, No. 2, p.127–38

Bird, D., Conlon, T., Swanson, J. (1996) Computing and information technology in Higher Still: Let's get it right *Scottish Educational Review* Volume 28, No. 1 p.3–15

Bloom, B.S. (1976) *Human Characteristics and School Learning* New York: MacGraw–Hill

Bloom, B.S. (1979) *Alterable Variables: The New Direction in Educational Research* Edinburgh: The Scottish Council for Research in Education

Boekaerts, M. (1988) Emotion, motivation and learning *International Journal of Educational Research* No. 12. p.229–34

Brandt, R.S. (1986) On Creativity and Thinking Skills: A Conversation with David Perkins *Educational Leadership*, May 1986, p.12–8

Brandt, R.S. (1990) On Knowledge and Cognitive Skills: A Conversation with David Perkins *Educational Leadership*, February 1990, p.50–3

Brandt, R.S. (1994) On Making Sense: A Conversation with Magdalene Lampert *Educational Leadership*, February 1994, p.26–30

Bransford, J.D., Vye, N.J. (1989) A Perspective on Cognitive Research and its Implications for Instruction in Resnick, L.B., Klopfer, L. E. (eds.) *Toward the Thinking Curriculum: Current Cognitive Research* Alexandria VA: ASCD

British Computer Society (1996) *A Glossary of Computing Terms eighth edition* London: Longman

Broadfoot, P. (1994) Exploring the Forgotten Continent: A Traveller's Tale *Scottish Educational Review* Vol. 26 No. 2

Brown, A.L., Palincsar, A.S. (1989) Guided Cooperative Learning and Individual Knowledge Acquisition in Resnick, L.B. (ed.) *Knowing, Learning and Instruction: Essays in Honor of Robert Glaser* Hillsdale, New Jersey: Lawrence Erlbaum Associates

Brown, G. (1984) Metacognition: New Insights into Old Problems? *British Journal of Educational Studies* Vol. XXXII No, 3 October, p.213–9

Brown, S. (1992) *Getting the Most from Observing Teaching SCRE Spotlight No. 32* Edinburgh: The Scottish Council for Research in Education

Brown, S., McIntyre, D. (1993) *Making Sense of Teaching* Buckingham: Open University Press

Choi, W.S., Repman, J. (1993) Effects of Pascal and FORTRAN Programming on the Problem-Solving Abilities of College Students *Journal of Research on Computing in Education* Vol. 25 No. 3 p.290–302

Clancy, M.J., Linn, M.C. (1992) *Designing Pascal Solutions: A Case Study Approach* New York: Computer Science Press

Clements, D.H., Gullo, D.F. (1984) Effects of Computer Programming on Young Children's Cognition *Journal of Educational Psychology* Vol. 76 No. 6 p.1051–8

Cole, A. *et al.* (1990) *Standard Grade Computing Studies: Proposals for a Course Revision* Unpublished paper Edinburgh: Moray House Institute

Collins, A., Brown, J.S., Newman, S.E. (1989) Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics in Resnick, L.B. (ed.) *Knowing, Learning and Instruction: Essays in Honor of Robert Glaser* Hillsdale, New York: Lawrence Erlbaum Associates

Coolican, H. (1994) *Research Methods and Statistics in Psychology 2nd edition* London: Hodder & Stoughton

Cope, P., Walsh, T. (1990) Programming in schools: 10 years on *Journal of Computer Assisted Learning* No. 6 p.119–27

Department of Education (1992) *Information Technology in Secondary Schools: A Review by HMI* London: HMSO

De Corte, E. (1990) Towards Powerful Learning Environments for the Acquisition of Problem-Solving Skills *European Journal of Psychology of Education* Vol V (1), p.5–19

diSessa, A.A., Abelson, H. (1986) Boxer: A Reconstructible Computational Medium *Communications of the ACM* Vol. 29, No. 9, p.859–68

diSessa, A.A. (1987) The Third Revolution in Computers and Education *Journal of Research in Science Teaching* Vol. 24 No. 4 p.343–67

Drever, E. (1985) Mastery Learning in Context, Theory and Practice in Brown, S., Munn, P. (eds.) *The Changing Face of Education 14 to 16: Curriculum and Assessment* Windsor: NFER–NELSON

Drever, E. (1988) Resource-Based Teaching: the New Pedagogy? in Brown, S., Wake, R. (eds.) *Education in Transition: What Role for Research?* Edinburgh: The Scottish Council for Research in Education

Drever, E. (1995) *Using Semi-Structured Interviews in Small-Scale Research A teacher's Guide* Edinburgh: The Scottish Council for Research in Education

Dromey, R.G. (1982) *How to Solve it by Computer* London: Prentice-Hall International

Dweck, C.S., Elliott, E.S. (1983) Achievement Motivation in Hetherington, E.M. (ed.) *Socialization, Personality and Social Development* (Vol. IV of Mussen, P.H. (ed.) *Handbook of Child Psychology*) New York: Wiley, p.643–92

Elliott, J. (1991) *Action Research for Educational Change* Milton Keynes: Open University Press

Ennis, D.L. (1994) Combining Problem-Solving Instruction and Programming Instruction to Increase the Problem-Solving Ability of High School Students *Journal of Research on Computing in Education* Vol. 26 No. 4 p.488– 96

Ennis, R.H. (1987) A Taxonomy of Critical Thinking Dispositions and Abilities in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Entwistle, N. J. (1987) Research on Motivation to Learn *Proceedings of the third meeting of the Forum on Educational Research Motivation in Education : What Role for Research:* Edinburgh: The Scottish Council for Research in Education

Flavell J, H. (1976) Metacognitive Aspects of Problem Solving in Resnick L. B. (ed.) *The Nature of Intelligence* Hillsdale NJ: Erlbaum

Flavell J.H. (1979) Metacognition and Cognitive Monitoring: A New Area of Cognitive–Developmental Inquiry *American Psychologist* Vol. 34, No. 10, p.906–11

Frederiksen, N. (1984) Implications of Cognitive Theory for Instruction in Problem Solving *Review of Educational Research* Vol. 54 (3), p.363–407

Fullan , M. (1993) *Change Forces: Probing the Depths of Educational Reform* London: The Falmer Press

Ganson, H., De Luca, C. (1996) Gender and Scottish Certificate of Education Presentations *Annual Proceedings of the Scottish Educational Research Association Conference September 1995*

Gardner, H. (1993) Educating for Understanding *The American School Board Journal* Vol. 180 (7), p.20–4

Gardner, H., Boix-Mansilla, V. (1994) Teaching for Understanding–Within and Across Disciplines *Educational Leadership* V. 51 (5) p.14–8

Goleman, D. (1996) *Emotional Intelligence* London: Bloomsbury

Goodyear, P. (1987) Sources of Difficulty in Assessing the Cognitive Effects of Learning to Program *Journal of Computer Assisted Learning* 3 p.214–23

Green, A. (1995) Verbal protocol analysis *The Psychologist* March, p.126–9

Greeno, J. G. (1980) Trends in the Theory of Knowledge for Problem Solving in Tuma, D. T., Reif, F. (eds.) *Problem Solving and Education: Issues in Teaching and Research* N.J.: Erlbaum

Higher Still Development Programme (1997) *Core Skills in Scottish Group Awards: Further Consultation Document* Autumn 1997

HM Inspectors of Schools (1993) *Effective Learning and Teaching in Scottish Secondary Schools Computing Studies* Edinburgh: The Scottish Office Education Department

HM Inspectors of Schools (1996) *Achievement for All* Edinburgh: The Scottish Office Education and Industry Department

Hadden R.A. (1991) *Problem Solving at the Bench: 100 Mini-Projects in Chemistry for 14–16 Year Olds* Glasgow: University of Glasgow

Harlen, W. (1994) *Evaluating Curricular Materials SCRE Spotlight No. 48* Edinburgh: The Scottish Council for Research in Education

Harlen, W., Malcolm, H. (1997) *Setting and Streaming: A Research Review* Edinburgh: The Scottish Council for Research in Education

Harvard University Project Zero (1992) *Enhancing Disciplinary Understanding in Teachers and Students Annual Report Submitted to the Spencer Foundation 1991–1992* (unpublished)

Hennessy, S. (1993) Situated Cognition and Cognitive Apprenticeship: Implications for Classroom Learning *Studies in Science Education,* Vol. 22, p.1–41

Holroyd, C. (1989) *Problem Solving and the Secondary Curriculum–A Review for the Scottish Education Department* (unpublished)

Husic, F.T., Linn, M.C., Sloane, K.D. (1989) Adapting Instruction to the Cognitive Demands of Learning to Program *Journal of Educational Psychology* Vol. 81 No. 4 p.570–83

Kafai, Y. B. (1993) *Minds in Play: Computer Game Design as a Context for Children's Learning* Ph.D. Thesis presented at the Graduate School of Education of Harvard University

Keller, J.K. (1990) Characteristics of Logo Instruction Promoting Transfer of Learning: A Research Review *Journal of Research on Computing in Education* Volume 23 No. 1 p.55–71

Kelly, A. (1989) Education or Indoctrination? The Ethics of School-Based Action Research in Burgess, G. (ed.) *The Ethics of Educational Research* Lewes, East Sussex: The Falmer Press

Kirkwood, M. (1986) *Programming in COMAL* Glasgow: Jordanhill Publications

Kirkwood, M. (1992) *Developing effective problem solving skills through programming* Paper presented at the Scottish Educational Research Association Conference at St Andrews September 1992 (unpublished)

Kirkwood, M. *et al.* (1992a) *Programming and Problem Solving in COMAL: Unit 1* Hamilton: Lanark Division EDS

Kirkwood, M. *et al.* (1993) *Programming and Problem Solving in COMAL: Units 2 & 3* Hamilton: Lanark Division EDS

Kirkwood, M. *et al.* (1994) *Programming and Problem Solving in COMAL: Unit 4* Hamilton: Lanark Division EDS

Kirkwood, M. (1995) *The Role of Metacognition, Higher Order Thinking and Problem Solving in Learning to Program* Paper presented at the European Conference on Educational Research Bath: September 1995 (unpublished)

Kirkwood, M. (1996) *Investigating Acceleration within an Individualized Learning Programme: A Study of Differentiation within Standard Grade Computing Studies* Report of an SOED funded project. Glasgow: University of Strathclyde

Kirkwood, M. (1997a) *Accelerating progress in learning* Summary Report of an SOED funded project. Glasgow: University of Strathclyde

Kirkwood, M. (1997b) Classroom Management in Secondary Schools in Bryce, T., Humes, W. (eds.) *Scottish Education* Edinburgh: Edinburgh University Press (in press)

Klawe, M., Levenson, N. (1995) Women in Computing: Where are we Now? *Communications of the ACM* Vol. 38, No. 1, p.29–35

Koffman, E. B. (1985) *Problem Solving and Structured Programming in Pascal* 2nd edition Reading, Massachusetts: Addison Wesley

Kurland, D.M., Pea, R.D., Clement, C., Mawby, R. (1986) *A Study of the Development of Programming Ability and Thinking Skills in High School Students* Bank Street College of Education New York

Lane and Lane (1986) Rationality, Self-esteem and Autonomy through Collaborative Enquiry *Oxford Review of Education* Vol. 12 No. 3 p.263–75)

Larkin, J.H. (1980) Teaching Problem Solving in Physics: The Psychological Laboratory and the Practical Classroom in Tuma, D.T., Reif, F. (eds.) *Problem Solving and Education: Issues in teaching and research* Hillsdale New Jersey: Lawrence Erlbaum Associates

Liao, Y., Bright, G.W. (1991) Effects of Computer Programming on Cognitive Outcomes: a Meta-Analysis *Journal of Educational Computing Research* Vol. 7 (3) p.251–68

Linn, M.C. (1985) The Cognitive Consequences of Programming Instruction in Classrooms *Educational Researcher* May p.14–16 & 25–29

Linn, M.C., Sloane, K.D., Clancy, M.J. (1987) Ideal and Actual Outcomes from Precollege Pascal Instruction *Journal of Research in Science Teaching* Vol. 24 No. 5 p.467–90

Linn, M. C., Clancy, M. J. (1992) The Case for Case Studies of Programming Problems *Communications of the ACM* Vol. 35 (3) p.121–32

Lipman, M. (1987) Some Thoughts on the Foundations of Reflective Education in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Martin, P. (1985) CRA or Certification? in *SCRE Newsletter 34* Edinburgh: The Scottish Council for Research in Education

Millar, R., Driver, R. (1987) Beyond Processes *Studies in Science Education* Vol. 14 p.33–62

Munn, P., Drever, E. (1990) *Using Questionnaires in Small-Scale Research* Edinburgh: The Scottish Council for Research in Education

Newell, A. (1980) One Final Word in Tuma, D.T., Reif, F. (eds.) *Problem Solving and Education: Issues in teaching and research* Hillsdale New Jersey: Lawrence Erlbaum Associates

Nickerson, R.S. (1987) Why teach thinking? in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Nisbet, J., Shucksmith, J. (1984) *The Seventh Sense* Edinburgh: The Scottish Council for Research in Education

Nisbet, J., Shucksmith, J. (1986) *Learning Strategies* London: Routledge Education Books

Nisbet, J. (1988) The Contribution of Research to Education in Brown, S., Wake, R. (eds.) *Education in Transition* Edinburgh: The Scottish Council for Research in Education

Nisbet, J. (1990) *Teaching Thinking: an Introduction to the Research Literature SCRE Spotlight No. 26* Edinburgh: The Scottish Council for Research in Education

Nisbet, J., Davies, P. (1990) The Curriculum Redefined: Learning to Think – Thinking to Learn *Research Papers in Education*, 5, 49–72

Otto, W. (1990) The Fickle Finger of Fats *Journal of Reading* January, p.290–3

Palincsar, A.S., Brown, A.L. (1989) Instruction for Self-Regulated Reading in Resnick, L.B., Klopfer, L.E. (eds.)*Toward the Thinking Curriculum: Current Cognitive Research* Alexandria VA: ASCD

Palumbo, D.B., Reed, W.M. (1991) The Effect of BASIC Programming Language Instruction on High School Students' Problem Solving Ability and Computer Anxiety *Journal of Research on Computing in Education* Vol. 23 No. 3 p.343–71

Papert, S. (1980) *MINDSTORMS: Children, Computers, and Powerful Ideas* Brighton, Sussex: Harvester Press Ltd.

Papert, S. (1994) *The Children's Machine – Rethinking School in the Age of the Computer* Hemel Hempstead, Hertfordshire: Harvester Wheatsheaf

Paul, R.W. (1987) Dialogical Thinking: Critical Thought Essential to the Acquisition of Rational Knowledge and Passions in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Pea, R.D., Kurland, D.M. (1984) *Logo Programming and the Development of Planning Skills* (Report No. 16) Bank Street College New York

Perkins, D. N., Martin, F. (1986) Fragile Knowledge and Neglected Strategies in Novice Programmers in Soloway, E., Iyengar, S. (eds.) *Empirical Studies of Programmers* Norwood, New Jersey: Ablex Publishing Corporation

Perkins D. N., Salomon, G. (1988) Teaching for Transfer *Educational Leadership* September p.22–32

Perkins D. N., Salomon, G. (1989) Are cognitive skills context bound? *Educational Researcher* Jan–Feb p.16–25

Perkins, D. (1993) Teaching for Understanding *American Educator* V.17 (3) p.8; p.28–35

Perkins, D., Blythe, T. (1994) Putting Understanding Up Front *Educational Leadership* V. 51 (5) p.4–7

Perrone, V. (1994) How to Engage Students in Learning *Educational Leadership* V. 51 (5) p.11–3

Pintrich, P.R., Berger, C.F., Stemmer, P.M. (1987) Students' Programming Behavior in a Pascal Course *Journal of Research in Science Teaching* Vol. 24 No. 5 p.451–66

Polya, G. (1948) *How to solve it* Princeton, New Jersey: Princeton University Press

Powney, J. (1996) The Complexities of Researching Values *Observations No. 8* Edinburgh: The Scottish Council for Research in Education

Resnick, L.B. (1987) *Education and learning to think* Washington D.C. : National Academic Press

Resnick, L.B., Klopfer, L. E. (1989) *Toward the Thinking Curriculum: Current Cognitive Research* Alexandria VA: ASCD

Salomon, G., Perkins, D.N. (1987) Transfer of Cognitive Skills from Programming: When and How? *Journal of Educational Computing Research* Vol 3 (2), p.149–69

Schoenfeld, A.H. (1985) *Mathematical Problem Solving* London: Academic Press

Schoenfeld, A.H. (1989) Teaching Mathematical Thinking and Problem Solving in Resnick, L.B., Klopfer, L. E. (eds.) *Toward the Thinking Curriculum: Current Cognitive Research* Alexandria VA: ASCD

Scottish CCC (1987) *Curriculum Design for the Secondary Stages* Dundee: Scottish CCC

Scottish Examination Board (1988) *Higher Grade Arrangements in Computing Studies* Dalkeith: SEB

Scottish Examination Board (1991) *Standard Grade Amended Arrangements in Computing Studies* Dalkeith: SEB

Scottish Examination Board (1993a) *Standard Grade Computing Studies Project Specifications–1995* Dalkeith: SEB

Scottish Examination Board (1993b) *Scottish Certificate of Education Computing Studies on the Standard Grade Guidance for teachers on the assessment of non-programming and programming coursework* Dalkeith: SEB

Scottish Examination Board (1996) *Examination Statistics 1995* Dalkeith: SEB

Scottish Office (1993) *Examination Results in Scottish Schools 1991–93* Edinburgh: Audit Unit HM Inspectors of Schools

Scottish Office (1995) *Examination Results in Scottish Schools 1993–95* Edinburgh: Audit Unit HM Inspectors of Schools

Scottish Office (1997) *Raising the Standard A White Paper on Educational Skills Development in Scotland: A Summary* Edinburgh: Scottish Office

Scriven, M. (1980) Prescriptive and Descriptive Approaches to Problem Solving in Tuma, D.T., Reif, F. (eds.) *Problem Solving and Education: Issues in teaching and research* Hillsdale New Jersey: Lawrence Erlbaum Associates

Simmons, R. (1994) The Horse Before the Cart: Assessing for Understanding *Educational Leadership* V. 51 (5) p.22–3

Simons, H. (1989) Ethics of Case Study in Educational Research and Evaluation in Burgess, G. (ed.) *The Ethics of Educational Research* Lewes, East Sussex: The Falmer Press

Simpson, M., Ure, J. (1993) *What's the Difference? – A Study of Differentiation in the Scottish Secondary School* Dundee: Northern College Publications

Simpson, M. (1997) Developing Differentiation Practices: Meeting the Needs of Pupils and Teachers *The Curriculum Journal* Vol. 8 No. 1 p.85–104

Singh, J.K. (1992) Cognitive Effects of Programming in Logo: A Review of Literature and Synthesis of Strategies for Research *Journal of Research on Computing in Education* Vol. 25 No. 1 p.88–104

Soloway, E., Ehrlich, K. (1984) Empirical Studies of Programming Knowledge *IEEE Transactions on Software Engineering* Vol. SE–10, No. 5, September p.595–609

Soloway, E. (1986) Learning to program = Learning to construct mechanisms and explanations *Communications of the ACM* Vol. 29 (9), p.850–8

Soloway, E. (1993) Should We Teach Students to Program? *Communications of the ACM* Vol. 36 No. 10 p.21–4

Spohrer, J.C., Soloway, E. (1986) Novice Mistakes: Are the Folk Wisdoms Correct? *Communications of the ACM* Vol. 29 No. 7 p.624–32

Stenhouse, L. (1980) Reflections in Stenhouse, L. (ed.) *Curriculum Research and Development* London: Heinemann Education

Stenhouse, L. (1983) Curriculum, Research and the Art of the Teacher in Stenhouse, L. (ed.) *Authority, Education and Emancipation: A Collection of Papers* London: Heinemann Education

Sternberg, R.J. (1987) Teaching Intelligence: The Application of Cognitive Psychology to the Improvement of Intellectual Skills in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Swartz, R.J. (1987) Teaching for Thinking: A Developmental Model for the Infusion of Thinking Skills into Mainstream Instruction in Baron, J.B., Sternberg, R.J. (eds.) *Teaching Thinking Skills: Theory and Practice* New York: W.H. Freeman & Company

Swartz, R. J., Parks, S. (1994) *Infusing the Teaching of Critical and Creative Thinking into Content Instruction* Pacific Grove: Critical Thinking Press & Software

Thomas, R. A., Upah, S., C. (1996) Give Programming Instruction a Chance *Journal of Research on Computing in Education* Volume 29 (1), p.96–108

Unger (1994) What Teaching for Understanding Looks Like *Educational Leadership* V. 51 (5) p.8–10

University of Glasgow (1997) *The Quality of Learning Report* 1996–97

Vygotsky, L.S. (1978) *Mind in Society. The Development of Higher Psychological Processes* Cambridge MA: Harvard University Press

Watt, J. (1995) *Evaluating Curricular Materials SCRE Spotlight No. 49* Edinburgh: The Scottish Council for Research in Education

Yoder, S., Moursund, D. (1991) Programming or Applications: How Best to Empower Students? *The Computing Teacher* November p.18–21

GLOSSARY OF COMPUTING AND PROGRAMMING TERMS [63]

Any term in bold indicates that an entry exists in the glossary which may help to clarify the definition.

algorithm

A sequence of steps designed to perform a particular task.

or a solution to a problem that is *independent* of any programming language.

array

A set of data items of the same type grouped together using a single identifier. A single column of data would be stored in a one dimensional array.

assignment

A statement in which a value is assigned to a **variable**. Two examples are:-

```
length:=20
area:= length * length
```

BASIC

A general purpose language that used to be available on most micro-computers. It was designed for beginners, and has a fairly simple operating environment.

branch

A branch is created by an **IF** or **CASE** statement in COMAL. By using a branch, the programmer can alter the order in which the program statements are obeyed (they are normally obeyed in sequence).

bug

A bug causes a **program** to perform incorrectly.

CASE statement

A case statement creates a multiple **branch** in COMAL. A simple example is:

```
CASE day OF
      WHEN 1, 21, 31
            ending$ := "st"
      WHEN 2, 22
            ending$ := "nd"
      WHEN 3, 23
            ending$ := "rd"
OTHERWISE
            ending$ := "th"
END CASE
```

The value of day when the statement is obeyed determines the outcome.

CLS

This COMAL command is used to clear the screen.

coding

This is the actual set of instructions which forms a **program**.

---

63 Several definitions are adapted from *British Computer Society* (1996) *A Glossary of Computing Terms eighth edition* London: Longmans

COMAL

COMAL is a general purpose language encouraging the writing of well structured programs. It was developed in Denmark during 1973/4, mainly for schools' use.

common algorithm

Also *standard algorithm.* Commonly encountered **algorithms** that have been developed to do specific tasks, such as to return the maximum value in a list.

condition

A condition can evaluate to either true or false. The condition in the statement, "IF score = 10 THEN PRINT "TOP MARKS!", is, "score = 10". Only if the condition is true, is the statement after THEN obeyed.

conditional loop

The commands inside the **loop** are repeated until a given **condition** is met (**REPEAT** loop), or only while a given condition holds true (WHILE loop).

constant

A fixed value which is not altered by the program.

contents

The value that a **variable** contains.

control structure

This term covers **branches** and **loops**, each of which affects the order in which the program is obeyed.

correct

The program produces the correct **output**.

data

Data is information which is in a form that can be processed by a computer system.

data type

See also **variable type**. A formal description of the kind of data being stored or manipulated within a program, for example, numerical data.

debug

A process of removing the **bugs** in a program.

design

The design is part of the **documentation** for the program. It consists of a list of ordered steps in English that the computer will obey when the program is run. See also **top-down design**.

documentation

This is a complete description of a program, intended for program maintenance. It will include (among other things) a statement of the purpose of the program, the **design**, **listings**, and **sample runs**.

file server

A file server provides central disk storage for any users of a **network**.

fixed loop

See **FOR loop**.

FOR loop

Also *fixed loop*. A type of loop where the number of repetitions is fixed in the program, e.g.

```
FOR counter := 1 TO 10 DO
      [loop body]
NEXT counter

FOR value := min TO max DO
      [loop body]
```

English language publications, keyword in title	1997 (Jan – Sep)	1992 – Sep 97
problem solving	21	784
thinking	30	1245
metacognition	1	42
understanding	38	1481
	90	3552

Table 1: Number of retrievals on keyword searches on the title of English language publications, using ERIC on the Internet, 4/12/97.

Australian, British, & Canadian publications: keyword in title	1976 – Sep 97 (Australian from 1978)
problem solving	1151
thinking	1379
metacognition	89
understanding	1797
	4416

Table 2: Number of retrievals on keyword searches on the title of publications, using International ERIC on CD-ROM, 4/12/97.

NEXT value

formatting — This refers mostly to the layout of text on the screen, and to the manner in which numbers are displayed (e.g. to 2 decimal places).

hand trace — This is a careful step-by-step simulation on paper of how an **algorithm** or **program** would be obeyed by the computer. It is used to verify that the algorithm or program is correct.

hardware — The physical part of a computer system.

high-level language — A high-level language is one designed to help a programmer express a computer program in a way that reflects the problem being solved, rather than the precise details of how the computer will produce the solution.

IF statement — An IF statement creates a **branch** in COMAL. A simple example is:

```
IF score = 10 THEN
    PRINT "Top marks!"
ELSE
    PRINT "You need more practice!"
END IF
```

The value of score when the statement is obeyed determines the outcome.

immediate mode — A command is entered (without a line number) followed by <RETURN>, and it is obeyed by the computer immediately.

inputs — The data values that are entered to the program.

identifier — The name that is chosen by the programmer for each **variable** or **procedure**. There are rules governing acceptable variable or procedure names, e.g. none should begin with a number or contain a space.

internal commentary — The programmer inserts comments in the program to help those reading the program to understand it and to modify it. In COMAL, each comment is inserted after `//` marks, and has no effect on the operation of the program, e.g. `VDU 2 // start printing`

library procedure — A library procedure is one which can be called up by the programmer to incorporate into a program. Each procedure in the library is designed to carry out a specific task.

listing — The **coding** is displayed on the screen or sent to the printer (printed listing).

logical error — This is an error in the logic of the program. Logical errors can be very subtle and difficult to uncover. They may go undetected if the program is not tested thoroughly and systematically.

loop — See also **FOR loop, REPEAT loop.** A loop causes a command, or sequence of commands, to be repeated when the program is obeyed.

loop counter

A variable that is specified in a **FOR** statement. Each time the loop is obeyed its value alters. In the example below, the loop counter is 'value'.

```
FOR value := min TO max DO
    [loop body]
NEXT value
```

macro

A sequence of instructions which are defined as a single element.

main program

The main program corresponds to the top-level of the **algorithm**. The execution of the program begins with the main program. It may include **calls** to **procedures**.

microworld

A 'place' where certain kinds of (mathematical or scientific or whatever...) thinking can hatch and grow with particular ease (Papert, 1980, p.125).

mode

The BBC micro-computer can display printing in several different ways by altering the screen mode. You can alter the number of rows, columns, and colours. Some modes support graphics and others do not.

modular

A modular design is one which is divided into separate sub-problems. A modular program is one which contains sub-programs (**procedures** and functions).

multimedia

The presentation of information by a computer system using graphics, animation, sound and text.

network

A number of computers are connected together in order to exchange information and to share access to facilities, such as a network printer. See also **file server, printer server.**

numeric variable

This refers to a **type** of **variable** on which calculations can be performed, e.g. in the statement below, 'length', 'breadth' and 'area' are all numeric variables:

```
area := length * breadth
```

outputs

The results that are produced by the program.

parameter

Information about a data item that is being supplied to a **procedure** when it is called.

pre-written procedure

The programmer can incorporate a pre-written procedure into a program. Two advantages of using such procedures are that it is an efficient way to produce a program, and any pre-written procedures should have been thoroughly tested in advance.

PRINT literal

The value to be **output** is a **constant** and not a **variable**, e.g.

```
PRINT 2              – to output the number 2
PRINT "HELLO"        – to output the word HELLO
```

printed listing

See **listing**.

printed output

The output from the program has been sent to the printer.

printer server

A printer server allows all the stations on a **network** to share the use of a printer.

procedure

A sub-program. The sequence of commands within the sub-program is encased between PROC and END PROC, e.g.

```
PROC show_menu
    [body of procedure]
END PROC show_menu
```

procedure call

A procedure call gets the computer to obey a **sub-program**. A procedure may be called from the **main program**, or (as shown below) from the body of another procedure.

```
PROC show_menu
    get_choices          --- procedure call
    display_choices      --- procedure call
END PROC show_menu
```

program

A list of commands or instructions to the computer in a programming language. Typing RUN will make the computer obey a COMAL program.

or an **algorithm** expressed in a programming language.

prompt

Message from a program that requests input from the user.

pseudocode

This is a method of describing a program which looks like a combination of English and a programming language.

READ .. DATA

In COMAL, READ is used to read DATA items from program lines. A simple example is:-

```
READ month$, day
DATA October, 31
```

This would have the effect of assigning the value 'October' to the **variable** month$, and a value of 31 to the variable day.

readable

A program that has been written in a style which makes it easy for another programmer to understand it and to modify it.

refinement

See also **top-down design** and **stepwise-refinement**. When a step (or section) is sub-divided, this results in a list of refinements of that step. A simple example, in which there are three refinements, is:-

```
refine "display table"
    1.  display headings for each column
    2.  for each row of the table
    3.          display day and rainfall amount
```

REPEAT loop

In COMAL, the command(s) inside the REPEAT **loop** is repeated until a given **condition** is met, e.g.

```
REPEAT
    READ name$
UNTIL name$ = "DAVID"
```

sample run — A sample run is obtained as evidence of testing the program. It is obtained by running the program using appropriate test data while the printer is enabled, so that all output to the screen is also sent to the printer.

semantics — The meaning of the individual elements which make up a computer language statement.

software — **Programs** which can be run on a computer system.

stepwise refinement — See **top-down design.**

string variable — This refers to a **type** of **variable** which is used to store text, e.g. in the statements below, "user_id$", and "password$" are string variables:

```
INPUT "Enter your user number": user_id$
INPUT "Enter your password": password$
```

In COMAL, a string variable is identified by the "$" symbol at the end of the name.

structured programming — A methodological approach to the design of a program which emphasizes breaking large and complex tasks into smaller sections.

syntax — The precise way program statements must be written to be understood by the computer.

syntax error — This type of error occurs when a program statement cannot be understood because it does not follow the rules laid down by the programming language.

TAB — A COMAL command which moves the cursor across, e.g. TAB(12) moves the cursor across 12 places from the left side of the screen, or to a particular column and row, e.g. TAB(10, 20) moves the cursor to column 10, row 20 (rows and columns are numbered from the top left of the screen).

test data — This is the data used to test the program. For a program the test data will show the **inputs** to the program and specify the expected **outputs.**

top-down design — Also **stepwise refinement.** A problem is defined very simply and then split into a number of smaller sections (or sub-problems). Each of these sections is successively split and refined until they are small enough to be programmed.

top-level — See **top-down design.** This is the first level of the design, before any refinements of steps are carried out.

type	See **data type**. A variable type describes the kind of data held by a variable. The two types of variable which are introduced on this course are **numeric variable** and **string variable**.
user documentation	User documentation to accompany a program is concerned with making the program easy to use.
user friendly	A system which is kind to its users.
user interface	The ways that have evolved for communication between the user and the computer.
validation	Computerised checking to detect any data that is unreasonable or incomplete. An example is a range check that can signal any data that is outwith a specified range.
variable	A variable is used whenever it is necessary to store a data value in the computer's memory. There is an identifier (or name) associated with each variable. The value of a variable can be altered by the program.
VDU 2, VDU 3	A VDU 2 command is used to send any subsequent output to the printer as well as to the screen. A VDU 3 command is used to revert to output being sent to the screen only.

A Study of Teaching and Learning in Computer Education:

Designing an introductory programming course to foster understanding, problem solving, higher order thinking and metacognition.

by

Margaret J. Kirkwood

A thesis submitted for
the degree of
Doctor of Philosophy
in the University of Glasgow

Volume 2

The Faculty of Arts, Department of Education
University of Glasgow
April 1998

List of Appendices

APPENDIX 1A CONTENT BY UNIT AND TOPIC

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 1 topic 1: Entering commands			
Programming language; the ability to create a range of effects using different commands; need for correct syntax; text.	Output only - of graphics, sound, text, a calculation.	Correct syntax errors; follow written instructions precisely; make accurate observations.	PRINT literal (to display text). CLS (to clear screen). Entering commands at the keyboard.
Unit 1 topic 2: Simple programs			
Stored program concept; line number conventions for simple programs; need for internal commentary at the start of a program; need to design solutions prior to coding.	Output only (of text).	Design a program by identifying and ordering the main solution steps; use a design grid to position text accurately on the screen; code the solution to correspond to the design; debug a program; consult reference material to aid debugging; test by running the program.	// (to indicate a comment). END PRINT (to display a blank line). LIST, RUN, NEW Enter, list, edit and run a program. Clear memory.
Unit 1 topic 3: Saving, loading & printing			
Save; load; program files; filenames & over-writing; list program files; output and printed output; printed listing.	Output only (of text).	Analyse a given problem statement, and design, implement and evaluate a solution with the aid of outline instructions. Analysis: describe the problem clearly and accurately. Design & implementation: as above. Evaluation: in terms of (i) its adequacy; (ii) improvements to the solution or solution method. Amend an existing solution to incorporate new features (working forward from the design).	VDU 2, VDU 3 (printer on, printer off). LOAD, SAVE, *CAT (to show a list of programs saved to disc). AUTO (to set up automatic line-numbering).

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 1 topic 4: Using coloured text			
Screen modes; foreground and background colours.	Output only (of text, in colour, and/or on a coloured background).	Experiment with different effects and tabulate the outcomes; anticipate and avoid problems (e.g. when background & foreground colours merge); plan strategically. Amend an existing solution to incorporate new features.	MODE (to alter screen mode). COLOUR (to select fore- or background colour). List part of a long program.
Unit 1 topic 5: Using procedures			
Sub-problems; sub-programs; main program; procedure; line number conventions for programs that include procedures; program commands are not always obeyed in sequence; readable programs; ease of amendment; proof of testing - sample run.	Output only (of text, within a sub-program).	Compare alternative solutions; produce a modular solution; provide evidence of testing; make predictions. Amend an existing solution to incorporate new features.	PROC, END PROC (to begin and end a procedure definition). Procedure call. Obtain a sample run.
Unit 2 topic 6: Input, process and output			
Input, process, output; use of pre-written procedures; use & benefits to the programmer of procedure libraries; user; data; input of text and input prompts (via pre-written procedures); sort words in alphabetical order (via pre-written procedures); run the same program with different data; choose suitable data to test the program; line number conventions for programs that include pre-written procedures; customize output.	Input, process, output.[1]	Analyse a given problem statement to identify the inputs, processes and outputs; use this analysis as a basis for designing the solution; devise suitable test data and test the program. Amend an existing solution to incorporate new features.	No new language features. Load set of pre-written procedures and then enter the main program. List the main program only, procedure only etc. Disable the printer while testing.
Unit 2 topic 7: Tabular output			
Input of text and numeric data; sort in text or numeric order; tabulate output (all via pre-written procedures).	As above.	Gather suitable test data by doing a survey in the class on a topic selected by the pupil. Amend an existing solution to incorporate new features.	TAB with semi-colon (to position text in columns). No new process skills.

1 All values are input and stored, all are processed, then output is produced (Clancy & Linn, 1992, p.390).

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 2 topic 8: Numeric variables			
Input of numeric data; simple calculation; output of the result. Numeric value; numeric variable; input a value to a numeric variable; arithmetical expression; assignment; expected and actual output; test data; output the value of a numeric variable. User-friendly features of programs (on-screen explanations, prompts for input, formatting of output). Readable programs use: meaningful variable names, internal commentary to explain each part of the program.	Prompt then input. As above.	Select suitable test data and calculate the expected output in advance. Compare the actual and expected output. Amend an existing solution to incorporate new, user-friendly features. Come up with an idea for a program which involves input of numeric values, a calculation and output. Solve this problem; ensure that the resulting program is readable and user-friendly.	INPUT variable name simple arithmetic expressions e.g. area:= length*breadth arithmetic operators:- + - * / ( ) PRINT variable name No new process skills
Unit 2 topic 9: String variables and loops			
Input and output of text; distinguish between output of a constant value and output of a variable. String value; string variable; you cannot use a string variable directly within a calculation; distinguish between numeric values and strings. Loop; fixed number of repetitions; loop counter. Idea that some of the steps of the design can be further sub-divided into smaller, more detailed steps, and a procedure can then be used to implement these steps.	Input, output. Do something a specified number of times.[2] Input, then loop to output a specified number of times. The number of repetitions is set (i) in the program; or (ii) by the user.	Compare programs to establish the differences between them. Solve a complex problem. Merge algorithms. Amend an existing solution to incorporate new, user-friendly features. Devise test data sets. Break down some of the steps from the design into smaller, more detailed steps. Match the program structure to the design, using procedures for any steps that have been sub-divided. The resulting program will be modular. Locate a logical error to within the main program or a particular procedure, narrow down the search.	INPUT string PRINT string FOR ... TO ... DO ... NEXT (fixed loop) No new process skills.

2 To perform some action(s) a specified number of times. In general, the easiest way to code this is a FOR loop (Clancy & Linn, 1992, p.399).

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 2 topic 10: Top-down design			
Top-down design process - idea of different levels of the design (top level, next level), refinement. Testing and debugging processes: debugging is time-consuming, difficult, some bugs do not generate any error messages, further bugs can easily be introduced. Careful design reduces the number of bugs; careful testing is essential to reveal hidden bugs.	As above. Also output the value of the loop counter repeatedly.	Tailor an existing design and program to solve a new problem. Debug a given program, explaining the strategy used to debug it.	No new language features. No new process skills.
Unit 3 topic 11: Conversion tables			
Identify the type of each input and assign a variable name, prior to designing the program. Format numbers as currency.	Input, then loop to process and output a specified number of times. Produce tabular output by displaying headings, then loop to process and output each row of the table in an appropriate format.	Obtain up-to-date information to use as test data. Merge algorithms. Tailor an existing design and program to solve a new problem, where there is some distance between the two. Modify a solution, experiment with new techniques and effects, apply ideas from the topic to a related problem, compare and evaluate different solutions.[3]	PRINT USING (to format numbers as currency).

3 These aspects are encompassed in the 'Ideas to Explore' section of each Unit 3 and Unit 4 topic.

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 3 topic 12: Nested loops			
Nest loops; trace a design or program by hand (using close tracking); purposes of close tracking; pass a parameter (by value).	Nested loops.	Close track a program, insert a (temporary) statement to make the computer wait for a key-press to see the effect of program statements, predict output in advance. Nest, tailor, merge algorithms. Amend an existing solution to incorporate new features. Generate alternative solutions and contrast them.	Formal and actual parameters. GET$ (to get a key press).
Unit 3 topic 13: Calculating a total			
Calculate an incremental total - initialize a total to zero and then accumulate its value; trace table; trace statement in program; read a value from a data statement & echo to the screen; produce user documentation.	Input one, process one.[4] Accumulate values until done.[5] Also read instead of input.	Merge algorithms. Abut one algorithm to another. Amend an existing solution to incorporate new features. Trace the value of a variable by inserting a trace statement in the program, construct a trace table by hand, and compare the outcomes of these two processes. Produce user documentation.	Assign a value to a numeric variable, e.g. total:= 0 Accumulate its value e.g. total := total + amount READ ... DATA (to read a value from a data statement).

4 Read a value, then process it, doing this over and over within a loop. The processing can include output (Clancy & Linn, 1992, p.398).

5 Accumulate a result from a variety of values e.g. add up values as they are input (Clancy & Linn, 1992, p.411).

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 3 topic 14: Keeping a count			
Count the occurrences of an event; generate a number at random; branch; conditional loop. Distinguish between a fixed loop and a conditional loop. Prompt for and validate input.	Count occurrences. Choose an action based on a given value. Get a value and process until done.[6] Prompt for and validate input.	Tailor, merge and nest algorithms. Generate alternative solutions and contrast them. Select between a fixed or conditional loop.	IF ... THEN ... (branch, contains single statement). REPEAT ... UNTIL ... (conditional loop) RND function (random number) conditional operators (=, <. >, <>, <=, >=)
Unit 4 topic 15: Branching			
Control structures for looping and branching; selecting from amongst these; logical operators. As part of the analysis of a problem, clarify any ambiguities in the problem statement, establish the boundaries of the problem and state your assumptions. Algorithm (in place of design); formal presentation of algorithm. Test report; choice of test data to represent typical and extreme situations.	Get a value and process until done. Select from alternatives.[7]	Analyse a problem statement in order to clarify any ambiguities that it contains, set out assumptions, and determine the boundaries of the problem. Produce a full test report. Merge and abut algorithms. Amend an existing solution to incorporate new features.	IF ... THEN ... END IF (contains multiple statements). IF ... THEN ... ELSE ... END IF (contains two options). CASE ... OF ... WHEN ... END CASE (contains multiple options). AND, OR (logical operators).
Unit 4 topic 16: Centred text			
Build up a library of procedures (saving each as a text file), & pass more than one parameter by value to a procedure. Use a driver program to test each procedure separately. Design a title screen, and centre text in a given position.	Read a value and process until done.	Tailor, merge and nest algorithms. Re-use elements of previous programs. Evaluate a solution in the following terms: (i) is it adequate? (ii) is it user-friendly? (iii) is it easy to amend? (iv) how can it be improved?	LEN function (returns length of string). DIV function (returns whole number on division). EOD function (detects end of data).

6 The processing can include output. This action may be coded with a repeat loop (Clancy & Linn, 1992, p.405).

7 Choose from alternative actions based on a given value (Clancy & Linn, 1992, p.400).

Programming concepts & principles introduced	Standard algorithms	Problem solving processes	Language features & process skills
Unit 4 topic 17: School concert			
Prompt for input (menu), and validate user's choice; concept of robustness; concept of an interactive program. Program a tune - sound channel, envelope, pitch, duration.	Prompt for and validate input. Select from alternatives.	From a (vague) statement of user requirements, define the problem clearly. Consider alternative solution methods. Read music on the stave (or find out how to do so, preferably from class-mates). Find ways to make the program more robust and user-friendly.	SOUND (produces a note). ENVELOPE (produces a sound envelope). CASE ... OF ... WHEN ... OTHERWISE ... END CASE
Unit 4 topic 18: Arrays			
One-dimensional array; array element; index; number of elements; dimensioning; filling an array and processing array elements.	Input, process, output. Fill an array.[8] Process every element of a one-dimensional array.[9] Select from alternatives.	Develop an alternative solution to that for Topic 13 and contrast the two. Merge, nest, tailor, abut algorithms (very complex problem which incorporates elements of many previous solutions).	DIM (to declare a one-dimensional array).

8 Read values from the input and store them in sequence in an array (Clancy & Linn, 1992, p.412).

9 Perform an action on each element of an array. Since the number of elements in an array is known, this is usually implemented with a for loop (Clancy & Linn, 1992, p.417).

APPENDIX 1B PROGRAMMING PROBLEMS, BY UNIT AND TOPIC

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 1 topic 2: Simple programs		
2.1 (Task 1)	First problem.	Examine a program which displays a letter shape (a 'T', three characters high), built from text characters, on the screen. *Full guidance and explanation.*
2.2 (Task 2)	Enhancement to 2.1	Amend the program to clear the screen first before the shape is displayed (a simple, modular design is presented). *Full guidance and explanation.*
2.3 (Tasks 3-4)	Second problem - similar to 2.2	Analyse a given problem statement, to design and write a program to display your own initials (five characters high) on the screen. Design, write and test a program to implement your solution. *Outline instructions only.*
Unit 1 topic 3: Saving, loading and printing		
3.1 (Tasks 1-2)	First problem	Examine a program which displays a person's name & address on the screen. Order given steps to produce the design for the program. *Full guidance and explanation.*
3.2 (Tasks 3-4)	Amendment to 3.1	Amend the program to display your own name and address (& save all programs from now on). *Full guidance and explanation.*
3.3 (Task 5)	Second problem - similar to 2.2 and 3.2 First assessment at foundation level.	Analyse a given problem statement, to design and write a program to display a party cracker type joke on the screen, with a border (e.g. x-x-x-x-x-x-x-x) above and below it. Design, write and test a program to implement your solution. Assess the adequacy of your solution. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*
3.4 (Tasks 7-10)	Enhancement to 3.2 and 3.3	Amend the designs and programs to obtain printed output. Also get a printed listing to document each solution. *Full guidance and explanation.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 1 topic 4: Using coloured text		
4.1 (Tasks 1-4)	First problem	Discover how to get different text and background colours in mode 1. *Some guidance and explanation.*
4.2 (Task 5)	Second problem - application of 4.1	Amend a program which displays a plain (white text on black background) version of the "Dennis the Menace" character, to one which displays his jumper with red and black hoops etc. *Some guidance.*
4.3 (Tasks 6-7)	Extension to 4.2	Amend the program to make the socks not matching (by changing the colours on one line). *Some explanation of the technique.*
Unit 1 topic 5: Using procedures		
5.1 (Tasks 1-5)	First problem	Analyse a given problem statement, to design and write a program to display four van shapes, one beneath the other, on the screen. Design, write and test a program which uses a 'van' procedure to implement a solution. *Full guidance and explanation, partial solution given.*
5.2 (Task 6)	Contrast with an alternative solution to 5.1	Compare the teacher's version of the program (which does not use any procedures) with your own. *Full guidance on which aspects to compare.*
5.3 (Task 7)	Amendment to 5.1	Amend the design and the program to display three vans, each in a different colour. *Outline instructions only.*
5.4 (Tasks 8-9)	Second problem - similar to 5.1	Analyse a given problem statement, to complete the design, and write a program to display your own ship design on the screen twice. Design your ship (using a design grid). Write the program to correspond to this list of main steps. Test the program (obtain a sample run). *Outline instructions only.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 1 topic 5: Using procedures		
5.5 (Task 10)	Third problem - similar to 5.3 and 5.4 Second assessment at foundation level.	Analyse a given problem statement, to design and write a program to display your own robot design twice, in different colours, on the screen. Design, write and test a program which uses a 'robot' procedure to implement your solution. Assess the adequacy of your solution. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*
Unit 2 topic 6: Input, process and output		
6.1 (Tasks 1-5)	First problem	Analyse a given problem statement, to design and write a program to input, sort alphabetically, and output to the screen and printer a list of your friends' first names. Design, write and test a program which uses pre-written procedures to implement your solution. *Full guidance and explanation, partial solution given.*
6.2 (Task 6)	Enhancement to 6.1	Amend the program to customize the output, so that the heading is "MY FRIENDS" instead of "WORDS". Test again. *Some guidance.*
6.3 (Task 7)	Second problem - similar to 6.2	Amend the program to alter the heading again (to "SCHOOL SUBJECTS"). Run the program with different test data - on school subjects studied. *Outline instructions only.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 2 topic 7: Tabular output		
7.1 (Tasks 1-2)	First problem	Analyse a given problem statement, to design and write a program to input, sort in descending order of votes, and output to the screen and printer in the form of a table, a list of television programmes and the votes cast for each. Design, write and test a program which uses pre-written procedures to implement your solution. *Information on what each pre-written procedure is designed to do, outline instructions only.*
7.2 (Task 3[10])	Second problem - similar to 7.1	Do your own survey in class - choose the topic. Amend the program to customize the headings for the table. Run the program with your test data. *Outline instructions only.*
7.3 (Task 4)	Third problem - extension to 7.1 Third assessment at foundation level.	Analyse a given problem statement, to design and write a program to output two lists - one in alphabetical order, the other in ascending numeric order (the data is on the populations of Scottish towns). Use suitable headings. Find out the population of five Scottish towns. Design, write and test a program which uses pre-written procedures to implement your solution. Assess the adequacy of your solution. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*

10 This problem can be omitted by any pupils who do not need it for consolidation.

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 2 topic 8: Numeric variables		
8.1 (Tasks 1-4)	First problem	Examine a program which calculates the area of a rectangle, given the length and breadth as inputs. *Full guidance and explanation, partial solution given.*
8.2 (Tasks 5-6)	Enhancement to 8.1	Assess how user-friendly the program is. Suggest three improvements to make the program more user-friendly. Compare this program with another version which incorporates a range of features to make it more user-friendly and readable. *Outline instructions only.*
8.3 (Tasks 7-9)	Second problem - similar to 8.1	Analyse a given problem statement, to design and write a program to calculate the average of two test scores. Design, write and test a program to implement your solution. Ensure that the program is both user-friendly and readable. *Outline instructions only, some assistance with the average formula.*
8.4 (Task 11)	Third problem - extension to 8.3	Create a problem statement of your own, which involves the computer doing a simple calculation.
	First assessment at general level.	Analyse your problem statement. Design, write and test a program to implement your solution. Ensure that the program is both user-friendly and readable. Assess the adequacy of your solution. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 2 topic 9: String variables and loops		
9.1 (Task 1)	First problem	Examine a program which inputs and outputs text. Enter your own name as data. Run the program again and enter a different name. *Outline instructions, explanation of new techniques.*
9.2 (Tasks 2-3)	Extension to 9.1	Compare this program with another version in which there is a fixed loop to display the text repeatedly. Amend the new version of the program to alter the number of repetitions from 10 to 20. *Outline instructions only.*
9.3 (Tasks 4-8)	Second problem - similar algorithm to 9.2	Analyse a given problem statement, to design and write a program to print a fixed number (set within the program) of posters for an event, the details of the event having been entered by the user. Design, write and test a program to implement your solution. *Outline instructions, explanation of top-down design process, partial solution given.*
9.4 (Task 9)	Enhancement to 9.3	Amend the design and program to enable the user to input the number of posters required. *Outline instructions, explanation of changes to the algorithm.*
Unit 2 topic 10: Top-down design		
10.1 (Tasks 1-2)	First problem, similar to 9.4[11]	Analyse a given problem statement, to design and write a program to print sets of address cards. Tailor an existing solution to solve the problem. *Outline instructions, further explanation of top-down design.*
10.2 (Tasks 3-4)	Second problem, extension to 10.1	Analyse a given problem statement, on printing numbered prize-draw tickets. Debug the given program. Describe how you detected each bug. *Outline instructions only.*

11 The coding of the solution can be omitted by any pupils who do not need consolidation of coding skills.

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 3 topic 11: Conversion tables		
11.1 (Tasks 1-6)	First problem	Analyse a given problem statement, to design and write a program to display a currency conversion table. The date, currency to convert to, country and exchange rate are entered by the user. Design, write and test a program to implement your solution. Use 'real' data to test the program. *Outline instructions, partial solution given, some explanation.*
11.2 (Task 7)	Enhancement to 11.1	Amend the program to format numbers as currency. *Outline instructions, explanation of technique.*
11.3 (Task 8)	Second problem - similar to 11.2 Second assessment at general level.	Analyse a given problem statement, to design and write a program to display a car rental charge table (for one day to fourteen days hire). The car model and daily rental charge are entered by the user. Tailor an existing solution to solve the problem. Assess the adequacy of your solution. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 3 topic 12: Nested loops		
12.1 (Task 1)	First problem	Given the design (which uses a loop), write a program to display five asterisks, one beneath the other. *Outline instructions only.*
12.2 (Task 2)	Second problem - similar to 12.1	Design and write a program to display five asterisks, this time side by side, using a loop. *Outline instructions, hint.*
12.3 (Tasks 3-7)	Third problem - combines the solutions to 12.1 and 12.2	Design and write a program to display a five by five array of asterisks. *Outline instructions, two hints.* Consider this version of the solution (design and coding) - note that there is one loop nested within another. Compare it with your solution. Consolidate your understanding of this program by (i) inserting a command for the computer to wait for a key-press before each asterisk is displayed, to see how the array is built up row by row; (ii) doing a hand trace; (iii) making predictions & testing these at the computer. *Outline instructions, explanation of new techniques.*
12.4 (Task 8)	Amendments to 12.3	Amend the given design and program to display a pattern of asterisks in the shape of - smaller square, rectangle, parallelogram, right-angled triangle. *Outline instructions, two hints.*
12.5 (Tasks 9-11)	Fourth problem - extension to 12.3	Analyse the pattern shown, comprising four squares of different dimensions (2 x 2, then 3 x 3 etc.) displayed one beneath the other. Design, write and test a program which uses a 'square' procedure, with one parameter, to display this pattern. *Outline instructions, partial solution, explanation of parameter passing.*
12.6 (Task 12)	Enhancement to 12.5	Amend the design and the program to use a loop within the main program which calls the 'square' procedure. *Outline instructions, hint.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 3 topic 13: Calculating a total		
13.1 (Tasks 1-6)	First problem	Analyse a given problem statement, to design and write a program to calculate the total rainfall for a given month based on the daily rainfall amounts which are entered to the program by the user. Make up suitable test data and: (i) calculate the expected output; (ii) load and run the program to compare the expected and actual output. Consolidate your understanding of the design and program by: (i) scrutinizing and comparing them; (ii) inserting a trace statement in the program (to echo the total to the screen after each new amount is added); (iii) making up a trace table; (iv) escaping the program (at a particular point) and displaying the current value of each variable; (v) ascertaining that the outcomes from (ii), (iii) and (iv) are identical; (vi) making predictions and testing these at the computer; (vii) giving an example of a running total being used in real life. *Outline instructions, explanations of new techniques, some partial solutions.*
13.2 (Task 7)	Second problem - extension to 13.1	Consider this new version of the design and program in which the rainfall amounts are stored within data statements and read by the program rather than being entered by the user. Load and test the program having first edited it to add the rainfall amounts as data. *Outline instructions, explanation of new technique.*
13.3 (Tasks 8-12)	Range of enhancements to 13.2 Third assessment at general level.	Amend the design and program to incorporate the following: (i) output a table with two headings, "Day" and "Rainfall amount"; (ii) allow the month, and the number of days in the month, to be read from data statements; (iii) calculate the average daily rainfall; (iv) produce user-documentation for the program (and find a 'real' user to test it - not someone from your programming class). *Outline instructions, explanations of new techniques.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 3 topic 14: Keeping a count		
14.1 (Task 1)	First problem	Follow instructions to simulate a random event on the computer by generating a random number (between 1 and 6) and echoing its value to the screen. Repeat the process several times to demonstrate that the result is random.
14.2 (Task 2)	Second problem - application of 14.1	Write a program to simulate 30 dice throws.
14.3 (Tasks 3-5)	Third problem - extension to 14.2	Examine this extension to the program, to count the number of sixes. Note the new features of the design and program. Consolidate your understanding of the design and program by: (i) loading the program and running it several times; (ii) amending the program to count the number of ones instead of sixes; (iii) experimenting with the use of different comparison operators e.g. count the results less than three. *Outline instructions, explanations of new techniques and language features.*
14.4 (Tasks 6-7)	Fourth problem - similar to 14.3	Analyse a given problem statement, to design and write a program to count the number of 'heads' and 'tails' for thirty simulated throws of a coin. Tailor an existing solution to solve the problem. Generate an alternative solution - either the two totals are accumulated, or one is calculated by subtracting the other from 30. Compare the two solutions. *Outline instructions.*
14.5 (Tasks 8-9)	Fifth problem - new algorithm	Analyse a given problem statement, to design and write a program to count the number of throws of a dice until a six is obtained. Examine the design, and do a hand trace. Write and test the program to correspond to the design. *Outline instructions, explanation of a conditional loop.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 3 topic 14: Keeping a count		
14.6 (Tasks 10-11)	Sixth problem - extension to 14.5	Analyse a given problem statement, to design and write a program to count the number of throws of two dice until a double is obtained. Design, write and test the program to implement the solution. Amend the solution to count the number of throws of two dice until a double six is obtained. *Outline instructions only.*
14.7 (Tasks 12-13)	Seventh problem - extension to 14.5	Analyse a given problem statement, to design and write a program to count the number of guesses taken by a player to find a 'hidden' number between 1 and 100. The program responds to each guess with either 'too high', 'too low', or the total number of guesses taken. Complete the design and the coding to obtain a solution. Experiment by pressing different keys when prompted to "Press the space-bar", and by replacing the 'next_screen' procedure (which includes validation) with CLS (which does not), to understand how the next_screen procedure works. *Outline instructions, some explanation of validation, partial solution.*
Unit 4 topic 15: Branching		
15.1 (Task 1)	First problem - enhancement to 14.4, and application of branching.	Read the summary of control structures for looping and branching. Select the appropriate control structure to use in order to implement the following enhancement (to 14.4), to display the word "head" or "tail" as each throw of the coin is made. *Outline instructions only.*
15.2 (Task 2)	Second (main) problem	Analyse a given problem statement, to design and write a program for a general knowledge quiz - five questions only. Complete the design and the coding to obtain a solution. Test the program systematically, in both typical and extreme situations. *Outline instructions, explanation of testing in typical and extreme situations.*
15.3 (Tasks 3-5)	Enhancements to 15.2	Amend the solution: (i) to accept upper- or lower-case responses; (ii) to vary the outputs depending on the final score. Document the final solution by producing a full test report. *Outline instructions, explanation of techniques.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 4 topic 16: Centred text		
16.1 (Tasks 1-3)	First problem	Derive a formula which can be used to position any text string (of less than 40 characters) in the middle of the current row. Test a given solution, which uses a driver program to call a "centre text" procedure, in which the text string is passed as a parameter to the procedure. Amend the driver program to accept the text string as an input. *Outline instructions, explanation of pre-defined functions and using a driver program.*
16.2 (Tasks 4-6)	Second problem - enhancement to 16.1	Amend the solution to position the text in any given row (e.g. ten lines from the top of the screen). Save each version of the procedure as a text file, to create a library of procedures that can be easily appended to any program. *Outline instructions, explanation of using two parameters and of saving text files.*
16.3 (Task 7)	Third problem - application of 16.2	Create a "title screen" procedure (displaying details of author, version number etc.) and save it to your procedure library. *Outline instructions.*
16.4 (Task 8)	Fourth problem - related to 16.1-16.3 First assessment at credit level.	Analyse a given problem statement, to design and write a program to produce a computer display for a school parents' night. The user presses the space-bar to see each screen-full of information, until all of it has been displayed. Design, code and test a solution which incorporates the following pre-written procedures: to centre text or centre text in a given row, to display a title screen, to see the next screen. Assess the adequacy of your solution. Appraise if your solution is user friendly and easy to adapt. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 4 topic 17: School concert		
17.1 (Tasks 1-5)	First problem	In the context of using a computer to help primary two pupils to practise their singing for a school concert ... Consider: (i) system requirements; (ii) is a computer solution appropriate, or would a tape of the music be better for pupils to use?; (iii) which (user-friendly) features should be incorporated into the solution? Appraise the suitability of a given input screen format which uses a menu to display a list of tunes to select from. Design your input screen format, and identify inputs and outputs. Examine how the computer produces sounds (run "tune1", "tune2", and "tune3" - get listings). Complete the coding for a fourth tune (you may need help to read the music) and save - with the other tunes - as a text file. Code and test the program based on the given algorithm, incorporating pre-written procedures as appropriate. Produce user documentation. *Outline instructions, explanation of using sound.*
17.2 (Tasks 6-8)	Enhancement to 17.1	Amend the design and coding to validate the user's choice from the menu. Suggest three further improvements to the solution. *Outline instructions, explanation of using a range check to validate the user's choice.*

Reference/ task(s)	Place in topic/link to other problems	Description of problem
Unit 4 topic 18: Arrays		
18.1 (Task 1)	First problem	In the context of the rainfall statistics problem (from Topic 13): Examine a given solution in which the rainfall amounts are read into an array and then output in the form of a table. *Outline instructions, explanation of one-dimensional arrays.*
18.2 (Task 2)	Second problem - extension to 18.1	Amend the coding (given the changes to the algorithm) to also compute the total rainfall and average daily rainfall. Contrast this solution, with that from Topic 13 (13.3). *Outline instructions.*
18.3 (Task 3)	Third problem - extension to 18.2 Second assessment at credit level.	In the context of the rainfall statistics problem: Analyse a given problem statement, to design and write a menu-driven program with the following display options: (i) a table, or (ii) bar graph, of the rainfall amounts; (iii) figures for the total and average; and (iv) a table showing the amount above or below average each day. Design, write and test a program to implement your solution. Assess the adequacy of your solution. Appraise if your solution is user friendly, easy to adapt and generalizable. Suggest some improvements to your solution and/or the solution method. *Outline instructions only.*

APPENDIX 1C PROGRAMMING SPECIFICATION: COMAL (MAIN ASPECTS BY UNIT)

Main Aspect	Unit	F/G/C	Unit	G/C	Unit	C
Representation Of Solution To Problem	1 1	*list of numbered steps, in English* *use grid to design output to screen*	2 2 3	*refine as list of steps* *identify problem inputs and outputs* as standard algorithm	3	*as the synthesis of elements of previous solutions*
Text Entry	1 1 2	*enter, use AUTO line numbering, edit, list on screen, obtain printed listing* *list part of program* list specified line number				
Sequence	1 1	*simple command* *procedure call*				
Modularity	1 2	*construction of simple procedure (PROC, END PROC)* use and adapt pre-written procedures			3	*value parameter*
Simple Data Types	1 2	*text* numeric				
Structured Data Types					4	one dimensional array (DIM)
Data Transfer	1 2 2	*output of text (PRINT)* input of numeric or string value (INPUT) output value of numeric or string variable (PRINT)	2 3 3	*format output (TAB and ;)* format numbers as currency (PRINT USING) use of READ ... DATA		
Statements/ Expressions	2	assignment involving simple arithmetic expression e.g. area:=length*breadth	2 3	*assignment involving more complex arithmetic expression e.g. average:=(score1+score2)/2* assignment of incremental expression (e.g. counter:=counter+1)	3 4 3 4	*pre-defined functions - RND, GET$,* also LEN, DIV, EOD *comparison operators* logical operators - AND, OR
Repetition			2 3	*fixed loop* loop counter used in loop	3 3	*nested control structures* *conditional loop (REPEAT ... UNTIL)*

Main Aspect	Unit F/G/C	Unit G/C	Unit C
Condition			3 *simple branch (IF ... THEN ...)* 4 other forms of IF statement 4 multiple branch (CASE statement)
Identify And Rectify Errors	1 *syntax, run-time and system errors*	2 *debug program*	3 *trace table* 3 *hand trace* 3 *trace statement*
Test Software	1 *run program, obtain printed output, obtain sample run* 2 use supplied test data	2 *select appropriate test data* 2 *comparison of expected and actual output* 2 *disable printer commands*	3 *hand trace* 4 generate full set of test data, to represent typical & extreme situations 4 produce test report
Readability	1 *internal commentary* 1 *choice of procedure name* 2 choice of variable name	2 *internal commentary to explain the main functions of the program*	
User Interface	1 *screen layout* 1 *screen mode* 1 *text and background colour*	2 *explain purpose to user* 2 *prompts for input* 2 *explain output to user* 3 produce tabular layout on screen 3 produce user documentation	3 *input validation* 4 use sound 4 present a menu of choices to the user 4 produce a title screen

APPENDIX 2A FOUR PROGRAMMING COURSEWORK ASSESSMENT TASKS & MARKING SCHEMES

Topic 11 Conversion tables

Task 8

Your answers to this task will go into your course work folder. So make sure that it is your own work, and do it carefully and neatly.

Collect a log sheet for this task from your teacher.

Also collect a MODE 7 design grid (one with the TAB positions marked on it).

*Read the **Problem Description** first then follow the instructions on the next page.*

Problem Description

Bargain Rent a Car have a range of car models for rent. The cost varies from £12 per day for the cheapest model to £60 per day for the dearest.

They ask you to write a program which outputs a table on the screen showing the cost of rental for between 1 and 14 days.

```
        Bargain Rent a Car

Model               Vauxhall Nova
Daily rental        £15.50

No. of days         Cost of hire (£s)
      1                   15.50
      2                   31.00
      ..                  ......
      14                  217.00
```

The user has to input the model, and the daily rental charge.

Topic 11 Conversion tables

Task 8

Use the log sheet to write down your answers.
Remember to fill in the details at the top.

1 Describe the problem clearly and accurately in your own words.

2 Identify the inputs to the program. Use a Mode 7 grid to design the output screen format.

3 Write down the top level of the design, and then refine each of the steps as necessary.

Use the design for the currency conversion problem to help you.

4 Write the program. Enter and test the program in stages. Correct any errors.

5 Save the program as ***rental***

6 Make up **two** sets of test data and obtain sample runs. Check that the output is correct in each case.

7 Obtain a printed listing.

8 Demonstrate the program to your teacher, and discuss how you will evaluate it.

9 Fill in the evaluation part of the log sheet.

10 Put these in your coursework folder - design grid, sample runs, printed listing and log sheet.

End of Topic 11

Tell your teacher that you have finished

Topic 13 Calculating a total

Outputting the data in tabular format

Your sample run from Task 7 shows only the monthly rainfall. To check that this figure is correct, you have to read the program listing to see the values that were read into the program as data. Hence you need to include the program listing and a sample run as evidence of testing.

You can overcome this problem by getting the program to output a table showing the amount of rainfall for each day, as well as outputting the monthly rainfall.

```
      Survey on rainfall - September
=======================================


          Day     Rainfall (mm)
           1           4
           2           1
           .           .
          30          12

     Monthly rainfall in mm     112
```

Note: You cannot see the whole table on the screen at once because the top part will scroll off the screen.

Task 8

1 Amend the design to do this.

Also use a Mode 1 design grid to work out the TAB positions accurately.

2 Edit the program (use different data to test it), and save it as ***survey4***

3 Get a printed listing and sample run. Check the output by doing a hand calculation.

Topic 13 Calculating a total

Extending the climate survey

You decide to extend the survey to obtain monthly rainfall figures for the next two months. You can then compare the figures from month to month.

```
     Survey on rainfall - October
=======================================


          Day     Rainfall (mm)
           1           5
           2           0
           .           .
          31          10

     Monthly rainfall in mm     136
```

```
     Survey on rainfall - November
=======================================


          Day     Rainfall (mm)
           1          12
           2           4
           .           .
          30           8

     Monthly rainfall in mm     98
```

Task 9

Edit the program to produce the output shown above for **October** (make up your own test data).

Save this version of the program as ***survey5*** and get a printed listing and sample run. Check the output by doing a hand calculation.

Topic 13 Calculating a total

Extending the climate survey - an alternative solution

The program can be amended so that only the **data** has to be changed in order to produce output for different months.

This can be done by supplying the month, and the number of days in the month, as data to the program.

```
210 END
215 DATA October, 31
220 // data on rainfall for each day
230 DATA 5,0,_,
```

These values are then read and assigned to appropriate variables in the main program,

```
READ month$, days_in_month
```

The variables replace the constant values in the program,

```
160 FOR day:=1 TO days_in_month DO

1030    PRINT TAB( 4); "Survey on rainfall -
        "; month$
```

Task 10

1 Update the design to include these changes.

Make the necessary changes to the program so that it produces output for the month of October (use the same test data as before).

Save the program as ***survey6*** and get a printed listing and sample run. Check that the output is the same as for Task 9.

Topic 13 Calculating a total

Task 10

2 Alter the data for the month of November.

Save the program as ***survey7*** and get a printed listing and sample run. Check the output by doing a hand calculation.

3 Contrast the different versions of the program (***survey5*** and ***survey6***). Which version is better, and why?

Outputting the average daily rainfall

One final addition to the program is to get it to calculate and output the **average daily rainfall**.

Note: You can calculate the average by dividing the total by the number of days in the month.

Task 11

1 Give a **full** and accurate description of the program in your own words.

2 Write out the design in full.

3 Edit the program (use the version which includes data for November).

Hint: Use `PRINT USING` to format the output for the average to 2 decimal places.

4 Save the program as ***survey8*** and get a printed listing and sample run. Check the output by doing a hand calculation.

Topic 13 Calculating a total

Task 12

In this task you will use word processing software to produce **user documentation** for the program which you developed in Task 11.

1 Use word processing software to produce a first draft of your user documentation. It should include each of the following:-

(i) a brief description of what the program does.

(ii) a description of the hardware and software requirements.

(iii) clear directions on how to load and run the program, and how to get a printed copy of the output.

2 Now get someone who is not in your Computing class to use the program with your user documentation.

Describe any problems that he or she had.

3 Make any editing changes that are necessary to improve your user documentation.

End of Topic 13

Tell your teacher that you have finished

Topic 14 Keeping a count

Task 11

Amend the design and the program so that the program ends when the first **double six** is thrown.

Save as ***dice7***

Guess the hidden number

This is a computer game in which the player has to guess the computer's hidden number.

```
        Guess the hidden number
        =======================


Try to guess the hidden number.  It lies
between 1 and 100.

You will be told if your guess is too
high or too low.

When you guess correctly you will be
told how many guesses you took.

Try to make as few guesses as possible.



          <Press the space-bar>
```

Screen 1

The hidden number is generated at random so that it is different each time the game is played.

Topic 14 Keeping a count

Task 12

1 Use Mode 1 design grids to plan the layout for:-

(i) screen 2, in which the player tries to guess the hidden number;

(ii) screen 3, which tells the player how many guesses he or she took.

2 The first four steps of the top-level design are shown below. Work out the remaining steps.

```
Design for "guess the hidden number"

Top-level
1  select mode 1
2  output the title
3  tell the player the rules
4  get the player to press the <space-
     bar> to see the next screen
```

3 Part of the program has already been coded. Load ***game*** and obtain a printed listing.

Finish coding the main program to match your design. (Save as ***game2***)

4 Try out your game. Get your friends to try it too.

5 Can you work out a strategy for playing the game to minimize the number of guesses?

Hint: Make your first guess 50.

Topic 14 Keeping a count

Validation of user input

The "guess the hidden number" program introduces a very important programming technique, one which you will use in nearly all of your programs from now on.

Within the `next_screen` procedure, there is a section of code which is used to validate the user's input.

```
3000 PROC next_screen
3010   // tells the player to press the
3020   // space-bar to see the next screen
3030   PRINT TAB(10, 30); "Press the
     <space-bar>"
3040   REPEAT
3050     key_press$:=GET$
3060   UNTIL key_press$ = " "
3070   CLS
3080 END PROC next_screen
```

The user must press the <space-bar> before the screen will clear. If he or she presses any other key (apart from the <ESCAPE> or <BREAK> keys), nothing will happen.

Task 13

1 Run the program. Try pressing a different key instead of the <space-bar>. What happens?

2 Change line 160 to a `CLS` command and run the program. Why is the version which uses `next_screen` better?

End of Topic 14

Tell your teacher that you have finished

Topic 16 Centred text

Task 8

Your answers to this task will go into your course work folder. So make sure that it is your own work, and do it carefully and neatly.

Collect a log sheet from your teacher, and a MODE 1 design grid.

*Read the **Problem Description** below and then follow the instructions on the next page.*

Design and write a program which will be useful for a school parents' night to help direct parents to the correct room for each teacher whom they have an appointment to see.

Here is an example screen. By pressing the <space-bar> repeatedly, the user can view the information on all of the teachers who will be there.

```
Aberness Grammar School
4th Year Parents' Night

      Miss Smith

        French

       Room 29


Press the <space-bar>
```

The program should read the data on each teacher - their name, subject, and location, from data statements. You should be able to include data for any number of teachers without altering the program.

Include a title screen at the start, and also a screen to welcome parents to the school. The text should be centred automatically on each line.

Topic 16 Centred text

Task 8

Either use the log sheet tor your answers, or word process them. Remember to put your details at the top. Include all documentation.

1 Describe the problem clearly and accurately in your own words.

2 Identify any inputs to the program. Design the output screen formats (Mode 1).

3 Identify any of your library procedures that would be useful to solve this problem.

4 Find out about the `EOD` command, and how you can use it with `REPEAT..UNTIL` to enable the program to read and process the data until the end of the data is reached.

5 Design the top-level algorithm and refine any steps as necessary.

6 Write the program, basing it on your algorithm.

7 Devise two sets of test data (restrict the number of teachers to a maximum of 5 in each set).

8 Enter and test the program (save as ***display***). Demonstrate the program to your teacher.

9 Evaluate the solution in the following terms.

(a) Does your program meet the specification?

(b) Is it user-friendly?

(c) Is the program written in a style that makes it easy to adapt?

(d) How could your solution be improved?

PROGRAMMING COURSEWORK TASK — CAR RENTAL: TOPIC 11 TASK 8

PUPIL'S NAME: **MARK:** **PERCENT:** **GRADE:**

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description. Log sheet: identification of problem inputs. Design grid: output format.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete. Problem inputs identified clearly. Problem outputs designed as appropriate.	2 2		
3	Log sheet: written design.	Outlines solutions. Refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	4		
4-8	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2 2		
9	Log sheet: written evaluation, two sample runs.	Assesses adequacy of solution. Suggests improvements.	• Accurate assessment. • Appropriate choice of test data, two sample runs. • Appropriate suggestion to improve the solution, solution method or original brief.	2 1		
			TOTAL	15		

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 6 MARKS
GRADE 5 (NARROW FAIL): 6 OR 7 MARKS
GRADE 4 (PASS): 8, 9, 10 OR 11 MARKS
GRADE 3 (GOOD PASS): 12 OR MORE MARKS

PROGRAMMING COURSEWORK TASK — RAINFALL DATA: TOPIC 13 TASKS 8-12

PUPIL'S NAME: MARK: PERCENT: GRADE:

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
Task 8 1	Design grid. Answer Booklet: written design.	Analyses problems. Outlines and refines solutions.	Designs output to the screen. Tailors design to incorporate: (i) output of headings; (ii) output of each row of the table.	3		
2-3	Printed listing. Sample run.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The program: • corresponds to the amended design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2		
		Assesses adequacy of solution.	Appropriate choice of test data, one sample run.	1 (6)		
Task 9	Printed listing. Sample run.	As above.	Identifies changes as (i) to the title; (ii) to the number of repetitions; and (iii) to the data. The amended program produces the correct output. Appropriate choice of test data, one sample run.	2 (2)		

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
Task 10 1-3	Answer Booklet: written design.	As above.	Identifies changes as: (i) the month and days in the month to be read from data statements; (ii) each to be represented by a variable. Tailors design accordingly.	2		
	Printed listing. Two sample runs.		The program: • corresponds to the amended design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2		
	Answer Booklet: written comparison.		Appropriate choice of test data, two sample runs. Contrast solutions (to Task 9 & Task 10). 2nd method less error-prone or takes less time to adapt the solution. Any other sensible reason.	2 (6)		

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
Task 11 1-2	Answer Booklet: written description & design.	As above.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	2		
			Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	2		
3-4	Printed listing. One sample run.		The program: • corresponds to the amended design; • is readable; • produces the correct output.	1		
			Appropriate choice of test data, one sample run.	1 (6)		
Task 12 1	Word processed text (draft).	Writes documentation.	As indicated in task.	3		
2	Answer Booklet: written evaluation.	Assesses adequacy.	Any problems identified clearly.	1		
3	Word processed text (final version).	Implements improvements.	Any necessary improvements have been made.	1 (5)		
			TOTAL	25		

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 11 MARKS
GRADE 5 (NARROW FAIL): 11-12 MARKS
GRADE 4 (PASS): 13-18 MARKS
GRADE 3 (GOOD PASS): 19 OR MORE MARKS

PUPIL'S NAME: **MARK:** **PERCENT:** **GRADE:**

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
Task 12 1-2	Design grids: output formats.	Analyses problems.	Problem outputs designed as appropriate.	2		
	Answer Booklet: written design.	Outlines and refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	4		
3-4	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	4		
5	Answer Booklet: written suggestion.	Discusses strategies.	The idea of a binary search or some other systematic search strategy.	1 (11)		
Task 13 1-2	Answer Booklet.	Assesses adequacy of solution.	• Accurate observation & comparison.	2		
	Supplementary evaluation sheet.	Assesses adequacy of solution.	• Accurate assessment in terms of (i) correctness; (ii) user-friendliness; (iii) adaptability.	3		
		Suggests improvements.	• Appropriate suggestion to improve the solution, solution method or original brief.	1 (6)		
			TOTAL	17		

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 7 MARKS
GRADE 3 (NARROW FAIL): 7-8 MARKS
GRADE 2 (PASS): 9-12 MARKS
GRADE 1 (GOOD PASS): 13 OR MORE MARKS

PUPIL'S NAME: **MARK:** **PERCENT:** **GRADE:**

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-3	Log sheet: written description, identification of problem inputs, library procedures. Design grids: output formats.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete. Problem inputs identified clearly. Problem outputs designed as appropriate. Useful library procedures identified.	2 4 1		
4-5	Log sheet: written design.	Outlines solutions. Refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	6		
6-8	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	4 2		
9	Log sheet: written evaluation, two sample runs.	Assesses adequacy of solution. Suggests improvements.	• Accurate assessment in terms of (i) correctness; (ii) user-friendliness; (iii) adaptability. • Appropriate choice of test data, two sample runs. • Appropriate suggestion to improve the solution, solution method or original brief.	3 2 1		
			TOTAL	25		

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 10 MARKS
GRADE 3 (NARROW FAIL): 10-12 MARKS
GRADE 2 (PASS): 13-18 MARKS
GRADE 1 (GOOD PASS): 19 OR MORE MARKS

PROGRAMMING COURSEWORK TASK CAR RENTAL: TOPIC 11 TASK 8

PUPIL'S NAME: ANTHONY **MARK:** 10 **PERCENT:** 67 **GRADE:** 4

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	2	2	*Accurate, reasonably complete (there could have been more information given on the format of the output), clear, and in his own words.*
	Log sheet: identification of problem inputs. Design grid: output format.		Problem inputs identified clearly. Problem outputs designed as appropriate.	2	0	*No separate identification of inputs. No output screen format.*
3	Log sheet: written design.	Outlines solutions. Refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	4	3	*Omitted to refine one step of top-level algorithm. Apart from this all other requirements are met.*
4-8	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output.	2	1	*Two minor errors in program, affecting layout and formatting of numbers.*
			There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2	2	*Satisfactory performance in these aspects (from teacher's assessment).*
9	Log sheet: written evaluation, two sample runs.	Assesses adequacy of solution.	• Accurate assessment. • Appropriate choice of test data, two sample runs.	2	1	*Only one test run, however valid choice of test data. Mentions a difficulty encountered.*
		Suggests improvements.	• Appropriate suggestion to improve the solution, solution method or original brief.	1	1	*Valid suggestion.*
			TOTAL	15	10	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 6 MARKS
GRADE 5 (NARROW FAIL): 6 OR 7 MARKS
GRADE 4 (PASS): 8, 9, 10 OR 11 MARKS
GRADE 3 (GOOD PASS): 12 OR MORE MARKS

PROGRAMMING COURSEWORK TASK CAR RENTAL: TOPIC 11 TASK 8

PUPIL'S NAME: GERARD **MARK:** 10 **PERCENT:** 67 **GRADE:** 4

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description. Log sheet: identification of problem inputs. Design grid: output format.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete. Problem inputs identified clearly. Problem outputs designed as appropriate.	2 2	2 1	*Incomplete - fails to mention one of the inputs. All other criteria met.* *No separate identification of inputs.*
3	Log sheet: written design.	Outlines solutions. Refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	4	3	*Omitted to refine one step of top-level algorithm.* *Apart from this all other requirements are met.*
4-8	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2 2	1 2	*One minor error in program, on formatting of numbers as currency. Some inaccuracies in internal commentary affect readability.* *Satisfactory performance in these aspects (from teacher's assessment).*
9	Log sheet: written evaluation, two sample runs.	Assesses adequacy of solution. Suggests improvements.	• Accurate assessment. • Appropriate choice of test data, two sample runs. • Appropriate suggestion to improve the solution, solution method or original brief.	2 1	1 0	*Valid choice of test data. Inaccurate assessment - no discussion.* *No suggestion made.*
			TOTAL	15	10	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 6 MARKS
GRADE 5 (NARROW FAIL): 6 OR 7 MARKS
GRADE 4 (PASS): 8, 9, 10 OR 11 MARKS
GRADE 3 (GOOD PASS): 12 OR MORE MARKS

PROGRAMMING COURSEWORK TASK CAR RENTAL: TOPIC 11 TASK 8

PUPIL'S NAME: DAWN **MARK:** 11 **PERCENT:** 73 **GRADE:** 4

Part(s)	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description. Log sheet: identification of problem inputs. Design grid: output format.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete. Problem inputs identified clearly. Problem outputs designed as appropriate.	2 2	1 2	*Wording very close to original wording, however she has extracted most of the relevant information.*
3	Log sheet: written design.	Outlines solutions. Refines solutions.	Design: • incorporates all of the required elements correctly; • is logically sequenced; • is modular; • is expressed clearly.	4	2	*Third input (number of days) wrongly introduced.* *Other inaccuracies resulting from incorrect tailoring of previous solution.* *Also some omissions.* *Sequencing logical, design modular, clearly stated.*
4-8	Printed listing, two sample runs.	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	The finished program: • corresponds to the design; • is readable; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2 2	1 2	*Correct implementation.* *Several inaccuracies in internal commentary affect readability.* *Satisfactory performance in these aspects (from teacher's assessment).*
9	Log sheet: written evaluation, two sample runs.	Assesses adequacy of solution. Suggests improvements.	• Accurate assessment. • Appropriate choice of test data, two sample runs. • Appropriate suggestion to improve the solution, solution method or original brief.	2 1	2 1	*Valid choice of test data.* *Mentions a problem with formatting numbers as currency.* *Suggests colour.*
			TOTAL	15	11	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 6 MARKS
GRADE 5 (NARROW FAIL): 6 OR 7 MARKS
GRADE 4 (PASS): 8, 9, 10 OR 11 MARKS
GRADE 3 (GOOD PASS): 12 OR MORE MARKS

APPENDIX 3A: CASE STUDY QUESTIONNAIRE FINDINGS (CLOSED ITEMS)

Unit 1:Closed items **Gender breakdown** **Attainment breakdown**

1. How interesting did you find the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
very interesting	3	15%	3	25%	0	0%	3	30%	0	0%
quite interesting	16	80%	9	75%	7	88%	7	70%	9	90%
quite boring	0	0%	0	0%	0	0%	0	0%	0	0%
very boring	1	5%	0	0%	1	12%	0	0%	1	10%
	20		12		8		10		10	

4. Were you able to understand what was taught ...?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
all of the time	5	25%	3	25%	2	25%	4	40%	1	10%
most of the time	12	60%	8	67%	4	50%	6	60%	6	60%
some of the time	2	10%	1	8%	1	12%	0	0%	2	20%
hardly any of the	1	5%	0	0%	1	12%	0	0%	1	10%
	20		12		8		10		10	

5. Were the instructions in the Task Sheets easy or difficult to follow?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
very easy	5	25%	4	33%	1	12%	3	30%	2	20%
quite easy	14	70%	8	67%	6	75%	7	70%	7	70%
quite difficult	0	0%	0	0%	0	0%	0	0%	0	0%
very difficult	1	5%	0	0%	1	12%	0	0%	1	10%
	20		12		8		10		10	

7. Did you like using the Answer Booklet?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	15	83%	11	100%	4	57%	8	89%	7	78%
no	3	17%	0	0%	3	43%	1	11%	2	22%
	18		11		7		9		9	

8. Were the Help Hints useful?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
very useful	2	22%	2	50%	0	0%	1	25%	1	20%
quite useful	5	56%	2	50%	3	60%	2	50%	3	60%
not useful	2	22%	0	0%	2	40%	1	25%	1	20%
	9		4		5		4		5	

Unit 1:Closed items **Gender breakdown** **Attainment breakdown**

9. Do you prefer to work ...?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
on your own	13	72%	8	80%	5	62%	8	100%	5	50%
with a partner	5	28%	2	20%	3	38%	0	0%	5	50%
	18		10		8		8		10	

10. Did you like working at your own pace?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	20	100%	12	100%	8	100%	10	100%	10	100%
no	0	0%	0	0%	0	0%	0	0%	0	0%
	20		12		8		10		10	

11. Were you able to get onto a computer whenever you needed to ...?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
all of the time	3	15%	1	8%	2	25%	0	0%	3	30%
most of the time	17	85%	11	92%	6	75%	10	100%	7	70%
only sometimes	0	0%	0	0%	0	0%	0	0%	0	0%
	20		12		8		10		10	

12. Had you ever done any computer programming before doing Unit 1?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	4	20%	2	17%	2	25%	2	20%	2	20%
no	16	80%	10	83%	6	75%	8	80%	8	80%
	20		12		8		10		10	

13. Are you looking forward to doing Unit 2?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	16	80%	11	92%	5	62%	9	90%	7	70%
no	1	5%	0	0%	1	12%	0	0%	1	10%
not sure	3	15%	1	8%	2	25%	1	10%	2	20%
	20		12		8		10		10	

Unit 1:Closed items **Gender breakdown** **Attainment breakdown**

6. Are you confident or not sure about each of these aspects?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. enter commands										
confident	18	90%	11	92%	7	88%	9	90%	9	90%
not sure	2	10%	1	8%	1	12%	1	10%	1	10%
	20		12		8		10		10	
b. enter program & edit										
confident	18	90%	12	100%	6	75%	10	100%	8	80%
not sure	2	10%	0	0%	2	25%	0	0%	2	20%
	20		12		8		10		10	
c. list & run program										
confident	19	95%	12	100%	7	88%	10	100%	9	90%
not sure	1	5%	0	0%	1	12%	0	0%	1	10%
	20		12		8		10		10	
d. save & load										
confident	20	100%	12	100%	8	100%	10	100%	10	100%
not sure	0	0%	0	0%	0	0%	0	0%	0	0%
	20		12		8		10		10	
e. get printed output										
confident	20	100%	12	100%	8	100%	10	100%	10	100%
not sure	0	0%	0	0%	0	0%	0	0%	0	0%
	20		12		8		10		10	
f. list design										
confident	16	80%	11	92%	5	62%	9	90%	7	70%
not sure	4	20%	1	8%	3	38%	1	10%	3	30%
	20		12		8		10		10	
g. use //, PRINT, END										
confident	19	95%	12	100%	7	88%	10	100%	9	90%
not sure	1	5%	0	0%	1	12%	0	0%	1	10%
	20		12		8		10		10	
h. use COLOUR										
confident	16	80%	12	100%	4	50%	10	100%	6	60%
not sure	4	20%	0	0%	4	50%	0	0%	4	40%
	20		12		8		10		10	
i. use PROC, END PROC										
confident	17	85%	12	100%	5	62%	10	100%	7	70%
not sure	3	15%	0	0%	3	38%	0	0%	3	30%
	20		12		8		10		10	

Unit 2:Closed items **Gender breakdown** **Attainment breakdown**

1. How interesting did you find the work in Unit 2 compared with Unit 1?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
more interesting	10	53%	8	67%	2	29%	6	60%	4	44%
about the same	8	42%	4	33%	4	57%	4	40%	4	44%
less interesting th	1	5%	0	0%	1	14%	0	0%	1	11%
	19		12		7		10		9	

4. Were you able to understand what was taught ...?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
all of the time	4	21%	4	33%	0	0%	4	40%	0	0%
most of the time	10	53%	5	42%	5	71%	6	60%	4	44%
some of the time	5	26%	3	25%	2	29%	0	0%	5	56%
hardly any of the	0	0%	0	0%	0	0%	0	0%	0	0%
	19		12		7		10		9	

5. Were the instructions in the Task Sheets easy or difficult to follow?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
very easy	4	21%	4	33%	0	0%	4	40%	0	0%
quite easy	12	63%	7	58%	5	71%	5	50%	7	78%
quite difficult	3	16%	1	8%	2	29%	1	10%	2	22%
very difficult	0	0%	0	0%	0	0%	0	0%	0	0%
	19		12		7		10		9	

8. Did the Revision Questions help you to do the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	14	74%	10	83%	4	57%	7	70%	7	78%
no	5	26%	2	17%	3	43%	3	30%	2	22%
	19		12		7		10		9	

9. Were the Summary Sheets useful to you?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	14	88%	10	91%	4	80%	8	100%	6	75%
no	2	12%	1	9%	1	20%	0	0%	2	25%
	16		11		5		8		8	

Unit 2:Closed items **Gender breakdown** **Attainment breakdown**

10. Are you pleased with your progress?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	10	53%	9	75%	1	14%	9	90%	1	11%
no	5	26%	1	8%	4	57%	1	10%	4	44%
not sure	4	21%	2	17%	2	29%	0	0%	4	44%
	19		12		7		10		9	

11. Are you looking forward to doing Unit 3?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	13	68%	9	75%	4	57%	9	90%	4	44%
no	3	16%	1	8%	2	29%	1	10%	2	22%
not sure	3	16%	2	17%	1	14%	0	0%	3	33%
	19		12		7		10		9	

Unit 2 Closed Items	Gender breakdown				Attainment breakdown			
6. Are you confident or not sure about each of these aspects?								
	All		**Girls**		**Boys**		**Credit**		**General**	
	N		N		N		N		N	
a. identify input-process-output										
confident	18	95%	11	92%	7	100%	10	100%	8	89%
not sure	1	5%	1	8%	0	0%	0	0%	1	11%
	19		12		7		10		9	
b. use pre-written procedures										
confident	16	84%	10	83%	6	86%	10	100%	6	67%
not sure	3	16%	2	17%	1	14%	0	0%	3	33%
	19		12		7		10		9	
c. test a program by entering different data each time										
confident	17	89%	11	92%	6	86%	10	100%	7	78%
not sure	2	11%	1	8%	1	14%	0	0%	2	22%
	19		12		7		10		9	
d. use numeric variables										
confident	12	63%	8	67%	4	57%	8	80%	4	44%
not sure	7	37%	4	33%	3	43%	2	20%	5	56%
	19		12		7		10		9	
e. calculate the expected & actual output when testing										
confident	16	84%	10	83%	6	86%	10	100%	6	67%
not sure	3	16%	2	17%	1	14%	0	0%	3	33%
	19		12		7		10		9	
f. write user-friendly programs										
confident	16	84%	9	75%	7	100%	10	100%	6	67%
not sure	3	16%	3	25%	0	0%	0	0%	3	33%
	19		12		7		10		9	
g. write readable programs										
confident	11	58%	8	67%	3	43%	7	70%	4	44%
not sure	8	42%	4	33%	4	57%	3	30%	5	56%
	19		12		7		10		9	
h. know which type of variable to use (numeric or string)										
confident	13	68%	8	67%	5	71%	7	70%	6	67%
not sure	6	32%	4	33%	2	29%	3	30%	3	33%
	19		12		7		10		9	
i. use a FOR loop										
confident	11	58%	7	58%	4	57%	8	80%	3	33%
not sure	8	42%	5	42%	3	43%	2	20%	6	67%
	19		12		7		10		9	
j. use top-down design										
confident	14	74%	10	83%	4	57%	7	70%	7	78%
not sure	5	26%	2	17%	3	43%	3	30%	2	22%
	19		12		7		10		9	
k. debug programs										
confident	15	79%	9	75%	6	86%	9	90%	6	67%
not sure	4	21%	3	25%	1	14%	1	10%	3	33%
	19		12		7		10		9	

Unit 3 Closed Items Gender breakdown Attainment breakdown

1. How interesting did you find the work in Unit 3 compared with Unit 2?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
more interesting	6	60%	5	62%	1	50%	6	60%	0
about the same	4	40%	3	38%	1	50%	4	40%	0
less interesting th	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0

4. Were you able to understand what was taught ...?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
all of the time	2	20%	2	25%	0	0%	2	20%	0
most of the time	8	80%	6	75%	2	100%	8	80%	0
some of the time	0	0%	0	0%	0	0%	0	0%	0
hardly any of the	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0

5. Were the instructions in the Task Sheets easy or difficult to follow?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
very easy	0	0%	0	0%	0	0%	0	0%	0
quite easy	8	80%	6	75%	2	100%	8	80%	0
quite difficult	2	20%	2	25%	0	0%	2	20%	0
very difficult	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0

8. Are you pleased with your progress?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
yes	8	80%	6	75%	2	100%	8	80%	0
no	0	0%	0	0%	0	0%	0	0%	0
not sure	2	20%	2	25%	0	0%	2	20%	0
	10		8		2		10		0

9. Did you try out any of the ideas to explore?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
yes	0	0%	0	0%	0	0%	0	0%	0
no	10	100%	8	100%	2	100%	10	100%	0
	10		8		2		10		0

10. Are you looking forward to doing Unit 4?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
yes	6	60%	6	75%	0	0%	6	60%	0
no	1	10%	0	0%	1	50%	1	10%	0
not sure	3	30%	2	25%	1	50%	3	30%	0
	10		8		2		10		0

Unit 3: Closed Items Gender breakdown Attainment breakdown

6. Are you confident or not sure about each of these aspects?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
a. use FOR loop to output a table									
confident	10	100%	8	100%	2	100%	10	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0
b. use TAB to align headings & columns									
confident	10	100%	8	100%	2	100%	10	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0
c. use one loop nested inside another									
confident	8	80%	6	75%	2	100%	8	80%	0
not sure	2	20%	2	25%	0	0%	2	20%	0
	10		8		2		10		0
d. use a parameter									
confident	4	40%	2	25%	2	100%	4	40%	0
not sure	6	60%	6	75%	0	0%	6	60%	0
	10		8		2		10		0
e. do a hand trace									
confident	8	80%	6	75%	2	100%	8	80%	0
not sure	2	20%	2	25%	0	0%	2	20%	0
	10		8		2		10		0
f. use a loop to calculate a total from input values									
confident	10	100%	8	100%	2	100%	10	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0
g. use a loop to calculate a total from data values									
confident	7	70%	6	75%	1	50%	7	70%	0
not sure	3	30%	2	25%	1	50%	3	30%	0
	10		8		2		10		0
h. make a trace table									
confident	8	80%	7	88%	1	50%	8	80%	0
not sure	2	20%	1	12%	1	50%	2	20%	0
	10		8		2		10		0
i. get the computer to trace the value of a variable									
confident	4	40%	4	50%	0	0%	4	40%	0
not sure	6	60%	4	50%	2	100%	6	60%	0
	10		8		2		10		0
j. use a counter & loop to count the occurrences of an event									
confident	9	100%	8	100%	1	100%	9	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	9		8		1		9		0
k. use a branch (IF ... THEN)									
confident	10	100%	8	100%	2	100%	10	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0

Unit 3: Closed Items Gender breakdown Attainment breakdown

6. Are you confident or not sure about each of these aspects?

	All		Girls		Boys		Credit		General
	N		N		N		N		N
l. use a conditional loop (REPEAT ... UNTIL)									
confident	10	100%	8	100%	2	100%	10	100%	0
not sure	0	0%	0	0%	0	0%	0	0%	0
	10		8		2		10		0

APPENDIX 3A: CASE STUDY QUESTIONNAIRE FINDINGS (OPEN ITEMS)

Unit 1: Open response items — Gender b/down — Attainment b/down

2. Which programs did you like the best? Say why you liked them.

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. think independently/use imagination	11	58%	7	58%	4	57%	6	60%	5	56%
b. sense of achievement/satisfaction	5	26%	3	25%	2	29%	3	30%	2	22%
c. learn & practice new techniques	7	37%	5	42%	2	29%	3	30%	4	44%
d. produce useful/realistic/relevant programs	0	0%	0	0%	0	0%	0	0%	0	0%
e. problem is easily done/understood	2	11%	2	17%	0	0%	2	20%	0	0%
f. problem is interesting/enjoyable/fun	2	11%	0	0%	2	29%	0	0%	2	22%
g. problem is difficult/challenging	1	5%	0	0%	1	14%	1	10%	0	0%
h. no reason given for stated preference(s)	0	0%	0	0%	0	0%	0	0%	0	0%
TOTAL RESPONDENTS	19		12		7		10		9	

3. Were there any things that you didn't like about the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. laborious/time-consuming	6	33%	4	36%	2	29%	4	44%	2	22%
b. "no", or a positive comment	6	33%	5	45%	1	14%	3	33%	3	33%
c. too repetitive	2	11%	0	0%	2	29%	0	0%	2	22%
d. too difficult/confusing	2	11%	1	9%	1	14%	1	11%	1	11%
e. an aspect of learning environment	2	11%	1	9%	1	14%	1	11%	1	11%
f. particular programs/techniques	0	0%	0	0%	0	0%	0	0%	0	0%
TOTAL RESPONDENTS	18		11		7		9		9	

7. Did you like using the Answer Booklet?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	15	83%	11	100%	4	57%	8	89%	7	78%
no	3	17%	0	0%	3	43%	1	11%	2	22%
	18		11		7		9		9	

Give a reason for your answer.

	All		Girls		Boys		Credit		General	
YES ...	N		N		N		N		N	
a. forms neat, well organized record	5	28%	4	36%	1	14%	4	44%	1	11%
b. is well designed & easy to use	4	22%	2	18%	2	29%	2	22%	2	22%
c. is good for referring back to	4	22%	3	27%	1	14%	2	22%	2	22%
d. helps you keep track, remember	4	22%	4	36%	0	0%	2	22%	2	22%
e. helps you understand work better	1	6%	1	9%	0	0%	0	0%	1	11%
f. cuts down on writing & saves time	1	6%	1	9%	0	0%	1	11%	0	0%
NO ...										
g. the format, as a separate booklet	2	11%	0	0%	2	29%	1	11%	1	11%
h. parts were hard to understand	1	6%	0	0%	1	14%	0	0%	1	11%
TOTAL RESPONDENTS	18		11		7		9		9	

Unit 1: Open response items **Gender b/down** **Attainment b/down**

9 Do you prefer to work on your own or with a partner?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
on your own	13	72%	8	80%	5	62%	8	100%	5	50%
with a partner	5	28%	2	20%	3	38%	0	0%	5	50%
	18		10		8		8		10	

Give a reason for your choice.

	All		Girls		Boys		Credit		General	
OWN ...	N		N		N		N		N	
a. can work at own pace	9	50%	5	50%	4	50%	6	75%	3	30%
b. fewer interruptions/distractions	4	22%	2	20%	2	25%	2	25%	2	20%
c. helps to self-monitor/aids understanding	4	22%	4	40%	0	0%	2	25%	2	20%
d. can be more independent/self-reliant	2	11%	2	20%	0	0%	2	25%	0	0%
e. gives more time on the computer	1	6%	1	10%	0	0%	0	0%	1	10%
PARTNER ...										
f. can help each other	2	11%	2	20%	0	0%	0	0%	2	20%
g. can share workload/ makes work easier	2	11%	1	10%	1	12%	0	0%	2	20%
h. can spot errors, easier to stay on track	2	11%	0	0%	2	25%	0	0%	2	20%
i. gives more time on the computer	1	6%	1	10%	0	0%	0	0%	1	10%
TOTAL RESPONDENTS	18		10		8		8		10	

Unit 2: Open response items **Gender b/down** **Attainment b/down**

2. Which programs did you like the best? Say why you liked them.

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. think independently/use imagination	5	26%	4	33%	1	14%	3	30%	2	22%
b. sense of achievement/satisfaction	3	16%	2	17%	1	14%	1	10%	2	22%
c. learn & practice new techniques	3	16%	3	25%	0	0%	2	20%	1	11%
d. produce useful/realistic/relevant programs	7	37%	3	25%	4	57%	3	30%	4	44%
e. problem is easily done/understood	2	11%	0	0%	2	29%	1	10%	1	11%
f. problem is interesting/enjoyable/fun	1	5%	1	8%	0	0%	1	10%	0	0%
g. problem is difficult/challenging	1	5%	1	8%	0	0%	1	10%	0	0%
h. no reason given for stated preference(s)	1	5%	1	8%	0	0%	1	10%	0	0%
TOTAL RESPONDENTS	19		12		7		10		9	

3. Were there any things that you didn't like about the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. laborious/time-consuming	6	35%	5	45%	1	17%	4	40%	2	29%
b. "no", or a positive comment	6	35%	4	36%	2	33%	3	30%	3	43%
c. too repetitive	3	18%	1	9%	2	33%	2	20%	1	14%
d. too difficult/confusing	1	6%	0	0%	1	17%	0	0%	1	14%
e. an aspect of learning environment	0	0%	0	0%	0	0%	0	0%	0	0%
f. particular programs/techniques	1	6%	1	9%	0	0%	1	10%	0	0%
TOTAL RESPONDENTS	17		11		6		10		7	

8. Did the Revision Questions help you to understand the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	14	74%	10	83%	4	57%	7	70%	7	78%
no	5	26%	2	17%	3	43%	3	30%	2	22%
	19		12		7		10		9	

Please explain your answer.

	All		Girls		Boys		Credit		General	
YES ...	N		N		N		N		N	
a. opportunity for more practice	7	37%	5	42%	2	29%	3	30%	4	44%
b. helped with retention	3	16%	3	25%	0	0%	3	30%	0	0%
c. helped to revise, test my knowledge	1	5%	1	8%	0	0%	1	10%	0	0%
d. no explanation given	5	26%	3	25%	2	29%	2	20%	3	33%
NO ...										
e. they didn't explain anything	1	5%	0	0%	1	14%	0	0%	1	11%
f. but they did help with retention	1	5%	1	8%	0	0%	1	10%	0	0%
g. no explanation given	3	16%	1	8%	2	29%	2	20%	1	11%
TOTAL RESPONDENTS	19		12		7		10		9	

Unit 2: Open response items **Gender b/down** **Attainment b/down**

9. Were the Summary Sheets useful to you?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
yes	14	88%	10	91%	4	80%	8	100%	6	75%
no	2	12%	1	9%	1	20%	0	0%	2	25%
	16		11		5		8		8	

Please explain your answer.

	All		Girls		Boys		Credit		General	
YES ...	N		N		N		N		N	
a. for revision & consolidation	7	44%	5	45%	2	40%	4	50%	3	38%
b. explained things well	5	31%	5	45%	0	0%	3	38%	2	25%
c. useful to refer to if I was stuck	3	19%	3	27%	0	0%	3	38%	0	0%
d. helped me remember, refreshed memory	2	12%	1	9%	1	20%	1	12%	1	12%
e. condensed the information well	2	12%	1	9%	1	20%	2	25%	0	0%
NO ...										
e. I didn't understand them	1	6%	1	9%	0	0%	0	0%	1	12%
f. no explanation given	1	6%	0	0%	1	20%	0	0%	1	12%
TOTAL RESPONDENTS	16		11		5		8		8	

Unit 3: Open response items — Gender b/down — Attainment b/down

2. Which programs did you like the best? Say why you liked them.

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. think independently/use imagination	0	0%	0	0%	0	0%	0	0%	0	0%
b. sense of achievement/satisfaction	6	60%	5	62%	1	50%	6	60%	0	0%
c. learn & practice new techniques	2	20%	2	25%	0	0%	2	20%	0	0%
d. produce useful/realistic/relevant programs	1	10%	1	12%	0	0%	1	10%	0	0%
e. problem is easily done/understood	2	20%	1	12%	1	50%	2	20%	0	0%
f. problem is interesting/enjoyable/fun	2	20%	2	25%	0	0%	2	20%	0	0%
g. problem is difficult/challenging	0	0%	0	0%	0	0%	0	0%	0	0%
h. no reason given for stated preference(s)	0	0%	0	0%	0	0%	0	0%	0	0%
TOTAL RESPONDENTS	10		8		2		10		0	

3. Were there any things that you didn't like about the work?

	All		Girls		Boys		Credit		General	
	N		N		N		N		N	
a. laborious/time-consuming	3	33%	3	43%	0	0%	3	33%	0	0%
b. "no", or a positive comment	2	22%	2	29%	0	0%	2	22%	0	0%
c. too repetitive	5	56%	3	43%	2	100%	5	56%	0	0%
d. too difficult/confusing	1	11%	1	14%	0	0%	1	11%	0	0%
e. an aspect of learning environment	0	0%	0	0%	0	0%	0	0%	0	0%
f. particular programs/techniques	0	0%	0	0%	0	0%	0	0%	0	0%
TOTAL RESPONDENTS	9		7		2		9		0	

ITEM	-	-	-	1.12	1.1	2.1	3.1	1.4	2.4	3.4	1.5	2.5	3.5	1.6, 2.6, 3.6
	Gender	*Programming coursework grade*	*Completed to end of...*	*Learned to program before doing U1?*	*How interesting, in U1?*	*How interesting, when compared with... U1?*	*U2?*	*Could you understand what was being taught in... U1?*	*U2?*	*U3?*	*Were the Task Sheet instructions easy/difficult to follow in... U1?*	*U2?*	*U3?*	*Confidence*
	(M/F)		(U1/U2/U3)	(Y/N)	(V/Q/QB/VB)	(M/S/L)		(A/M/S/HA)			(VE/QE/QD/VD)			% "confident"
Bryan	M	4	U2	N	quite inter.	same	*	most	most	*	QE	QE	*	75%
Stephen	M	4	U1	N	very boring	*	*	hardly	*	*	VD	*	*	33%
Claire	F	3	U2	Y	quite inter.	more	*	most	some	*	QE	QE	*	75%
Scott	M	4	U2	N	quite inter.	same	*	most	some	*	QE	QE	*	80%
Paula	F	1	U3	N	very inter.	more	more	all	all	all	QE	VE	QE	97%
Claudia	F	2	U3	N	quite inter.	more	same	most	most	most	QE	QE	QD	88%
Carol-Anne	F	3	U2	N	quite inter.	same	*	most	most	*	VE	QE	*	100%
Dawn	F	4	U2	N	quite inter.	more	*	most	some	*	QE	QE	*	45%
Chris	M	1	U3	N	quite inter.	more	same	most	most	most	QE	QE	QE	84%
Caroline	F	1	U3	N	very inter.	more	more	most	all	most	VE	QE	QE	81%
Anthony	M	4	U2	N	quite inter.	more	*	most	most	*	QE	QE	*	70%
Michael	M	4	U2	Y	quite inter.	less	*	all	most	*	VE	QE	*	100%
Kirsty	F	1	U3	N	quite inter.	more	more	all	most	all	VE	VE	QE	84%
Bernadette	F	3	U2	N	quite inter.	more	*	some	some	*	QE	QD	*	75%
Kelly	F	2	U3	N	quite inter.	more	more	most	most	most	QE	VE	QE	80%
Catherine	F	2	U3	N	quite inter.	same	more	most	most	most	QE	QE	QE	94%
Kathleen	F	2	U3	Y	quite inter.	same	same	all	all	most	VE	VE	QD	91%
David	M	2	U3	Y	quite inter.	same	more	all	most	most	QE	QD	QE	88%
Gerard	M	4	U2	N	quite inter.	same	*	some	some	*	QE	QD	*	55%
Lorraine	F	1	U3	N	very inter.	same	same	most	all	most	QE	QE	QE	94%

KEY

* - not applicable

V - very
Q - quite
QB - quite boring
VB - very boring

M - more
S - about the same
L - less

A - all of the time
M - most ...
S - some ...
HA - hardly any ...

VE - very easy
QE - quite ...
QD - quite difficult
VD - very ...

APPENDIX 3B: PROFILE OF INDIVIDUAL RESPONSES TO CLOSED ITEMS IN QUESTIONNAIRES

ITEM	1.11	1.8	1.7	2.8	2.9	1.10	1.9	2.10	3.8	1.13	2.11	3.10
	Could you get access to a computer?	*Help Hints useful?*	*Did you like the Answer Booklet?*	*Revision Questions helpful?*	*Summary Sheets useful?*	*Do you like working at your own pace?*	*On own/ with partner?*	*Satisfaction with progress by end of .. U2?*	*U3?*	*Are you looking forward to doing ... U2?*	*U3?*	*U4?*
	(A/M/S)	(V/Q/N)	(Y/N)	(Y/N)	(Y/N)	(Y/N)	(O/P)	(Y/N/U)		(Y/N/U)		
Bryan	most	quite useful	Y	N		Y	own	N	*	Y	Y	*
Stephen	all	not useful	N	*	*	Y	partner	*	*	N	*	*
Claire	most		Y	Y	Y	Y	partner	U	*	Y	U	*
Scott	most			N	Y	Y	own	U	*	U	U	*
Paula	most	very useful	Y	Y	Y	Y	own	Y	U	Y	Y	Y
Claudia	most	quite useful	Y	Y	Y	Y	own	Y	Y	Y	Y	Y
Carol-Anne	all		Y	Y	Y	Y	own	Y	*	Y	Y	*
Dawn	most	very useful	Y	Y	Y	Y	partner	U	*	Y	U	*
Chris	most	not useful	Y	N	Y	Y	own	Y	Y	Y	Y	U
Caroline	most	quite useful	Y	Y	Y	Y	own	Y	U	Y	Y	U
Anthony	most	quite useful	N	Y	Y	Y	own	U	*	Y	Y	*
Michael	most	quite useful	Y	Y	Y	Y	partner	N	*	Y	Y	*
Kirsty	most			Y	Y	Y	own	Y	Y	Y	Y	Y
Bernadette	most		Y	Y	N	Y	own	N	*	Y	N	*
Kelly	most		Y	Y	Y	Y	[both]	Y	Y	Y	Y	Y
Catherine	most		Y	N		Y	[both]	Y	Y	U	Y	Y
Kathleen	most		Y	Y	Y	Y	own	Y	Y	Y	Y	Y
David	most		N	Y		Y	own	N	Y	Y	N	N
Gerard	all		Y	Y	N	Y	partner	N	*	U	N	*
Lorraine	most		Y	N	Y	Y	own	Y	Y	Y	Y	U

KEY

A - all of the time
M - most ...
S - only sometimes

* - not applicable

Y - yes
N - no
U - uncertain

Y - yes
N - no
U - uncertain

THREE EXAMPLES OF HOW I APPLIED THE MARKING CRITERIA TO PUPILS' SPREADSHEET WORK (COURSEWORK ITEM)

PUPIL'S NAME: Anthony **MARK: 10.5** **PERCENT: 44%** **GRADE: 5**

CORE (GENERAL LEVEL)

Tasks	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	4	1	*Not clear, some aspects omitted, some inaccuracies (e.g. mention of bar chart and graph as output).*
	Log sheet: identification of resources.		Hardware and software resources identified correctly.	2	1.5	*Spreadsheet package not named.*
3	Log sheet: identification of formulae.	Outlines solutions. Refines solutions.	The correct formula identified for: • total • average • maximum • minimum	4	3	*Wrong column for one set of max/min formula.*
4-6	Printed output (spreadsheet, graph).	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	Enter spreadsheet details correctly. Print spreadsheet & contents. Print bar chart.	2 2 2	2 1 0	*Correct.* *No contents printed.* *No bar chart.*
			The finished spreadsheet: • corresponds to the design; • produces the correct output.	2	2	*Correct.*
			There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2	0	*No items checked on list of things to be included (nothing entered on back of log sheet at all).*
7-8	Log sheet: written evaluation.	Considers implications.	• Explanation of how the spreadsheet could be adapted to cover a different year.	2	0	*Hasn't answered.*
		Assesses adequacy of solution.	• Accurate assessment (with explanation).	2	0	*Hasn't answered.*
			TOTAL	24	10.5	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 10 MARKS
GRADE 5 (NARROW FAIL): 10 or 11 MARKS
GRADE 4 (PASS): 12 - 17 MARKS
GRADE 3 (GOOD PASS): 18 OR MORE MARKS

THREE EXAMPLES OF HOW I APPLIED THE MARKING CRITERIA TO PUPILS' SPREADSHEET WORK (COURSEWORK ITEM)

PUPIL'S NAME: Gerard **MARK: 22/24** **PERCENT: 92%** **GRADE: 3**

CORE (GENERAL LEVEL)

Tasks	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	4	3	*Description could have been clearer e.g. no mention of whether statistics refer to month or year.*
	Log sheet: identification of resources.		Hardware and software resources identified correctly.	2	2	
3	Log sheet: identification of formulae.	Outlines solutions. Refines solutions.	The correct formula identified for: • total • average • maximum • minimum	4	4	
4-6	Printed output (spreadsheet, graph).	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	Enter spreadsheet details correctly. Print spreadsheet & contents. Print bar chart. The finished spreadsheet: • corresponds to the design; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2 2 2 2 2	2 2 2 2 2	*Correct.* *Included.* *Included & correct.* *Correct.* *All items checked and included.*
7-8	Log sheet: written evaluation.	Considers implications. Assesses adequacy of solution.	• Explanation of how the spreadsheet could be adapted to cover a different year. • Accurate assessment (with explanation).	2 2	2 1	*Satisfactory (if not entirely full) explanation.* *Accurate but no explanation.*
			TOTAL	24	22	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 10 MARKS
GRADE 5 (NARROW FAIL): 10 or 11 MARKS
GRADE 4 (PASS): 12 - 17 MARKS
GRADE 3 (GOOD PASS): 18 OR MORE MARKS

THREE EXAMPLES OF HOW I APPLIED THE MARKING CRITERIA TO PUPILS' SPREADSHEET WORK (COURSEWORK ITEM)

PUPIL'S NAME: Dawn **MARK: 20/24** **PERCENT: 83%** **GRADE: 3**

CORE (GENERAL LEVEL)

Tasks	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1-2	Log sheet: written description.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	4	2	*Rather unclear and incomplete - doesn't explain the nature of the data.*
	Log sheet: identification of resources.		Hardware and software resources identified correctly.	2	1	*Does not name spreadsheet or computer.*
3	Log sheet: identification of formulae.	Outlines solutions. Refines solutions.	The correct formula identified for: • total • average • maximum • minimum	4	4	
4-6	Printed output (spreadsheet, graph).	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	Enter spreadsheet details correctly. Print spreadsheet & contents. Print bar chart. The finished spreadsheet: • corresponds to the design; • produces the correct output. There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2 2 2 2 2	2 2 2 2 2	*Correct.* *Included.* *Included & correct.* *Correct.* *All items checked and included.*
7-8	Log sheet: written evaluation.	Considers implications.	• Explanation of how the spreadsheet could be adapted to cover a different year.	2	2	*Good explanation.*
		Assesses adequacy of solution.	• Accurate assessment (with explanation).	2	1	*Accurate but no explanation.*
			TOTAL	24	20	

CUT-OFF SCORES:

GRADE 7 (FAIL): LESS THAN 10 MARKS
GRADE 5 (NARROW FAIL): 10 or 11 MARKS
GRADE 4 (PASS): 12 - 17 MARKS
GRADE 3 (GOOD PASS): 18 OR MORE MARKS

THREE EXAMPLES OF HOW I APPLIED THE MARKING CRITERIA TO PUPILS' SPREADSHEET WORK (COURSEWORK ITEM)

PUPIL'S NAME: Dawn **MARK: 17.5/22** **PERCENT: 80%** **GRADE: 1**

EXTENSION (CREDIT)

Tasks	Evidence	PA	Criteria	Possible marks	Actual marks	Comment
1	Log sheet: written description.	Analyses problems.	Description: • in pupil's own words; • clearly stated; • accurate; • complete.	2	1	*Wording needs to be more precise.*
2-3	Design grid and Log sheet: identification of formulae.	Outlines solutions. Refines solutions.	The correct formula identified for: • difference • sunny/dull • other amounts (total, average, min, max)	3	3	*Works out the difference twice, but values are correct.*
			Identify formulae to be replicated & whether absolute/relative.	4	1.5	*Three formulae identified with absolute reference indicated in formula.*
4-5	Printed output (spreadsheet).	Uses hardware & software tools effectively & efficiently. Identifies & rectifies errors. Discusses progress.	Edit spreadsheet to add columns & formulae.	2	2	*Done correctly.*
			Print the spreadsheet & its contents.	2	2	*Included.*
			The finished spreadsheet: • corresponds to the design; • produces the correct output.	2	2	*Meets requirements.*
			There is evidence of: • planning & organization; • on-going monitoring & testing; • the ability to discuss progress.	2	2	*All items checked and included.* *Documentation carefully presented.* *No help sought.*
6-9	Log sheet: written evaluation.	Considers implications.	• Identification of need to lock cells to safeguard information, or other appropriate suggestion.	1	1	*Clear explanation.*
		Assesses adequacy.	• Accurate assessment.	2	1	*Accurate but no reasons given.*
		Suggests improvement.	• Any suitable suggestion.	1	1	
		Considers implications.	• Suggestion for adapting s/sheet.	1	1	
			TOTAL	22	17.5	

CUT-OFF SCORES: GRADE 2 (PASS): 11 - 16 MARKS
GRADE 1 (GOOD PASS): 17 OR MORE MARKS

APPENDIX 5A REVIEW TASK 3 – THEMATIC ANALYSIS

Note: The number in brackets is the number of responses.

Learn to use the keyboard [5]

Take your time to learn where the keys are on the keyboard. This makes your keying in faster for long programs.

Learn to use other computers [3]

Try to get a chance to use all types of computers.

Password [9]

Do not tell anyone your password. Because some people will not hesitate in going into your user area and changing or deleting programs.

Don't change your password too often or you will forget what you have changed it to. Mrs xxx will go mad and not let you on the computer.

Saving and printing [17]

Always save your tasks as the name given in the task sheets. This makes it easier when trying to find programs.

Save everything every so often so that you don't lose any work.

Don't hog the printer or Mrs xxx will not let you onto the computers.

Always load in the printer driver as soon as you log on. This saves getting annoyed.

Keep your work organized [16]

Make sure you write the topic and task numbers at the top of your printed listings and outputs. And staple them together after every topic. This will be quicker for both you and the teacher when she comes to correct them.

Organize your printouts as you go along and don't let them pile up. This will make it easier to refer back to your previous examples and will save you time.

Maintain your progress diary [7]

Keep your diary up to date so you can remember what you done the last day, and can ask the teacher if you had difficulty with any of the previous work.

Plan what you are going to do [2]

Plan the steps in your head so that you have a clear idea of what's going on. This will also save time. Make sure your plan of action works, write your ideas down.

Get your work checked by the teacher [3]

Get your work corrected so you know if you've made any mistakes and you can then go back and correct any errors.

Doing extra work [6]

If you have been off or missed any work, come up at lunchtimes, as it can be easy to fall back on the work.

Remember if you take things home always bring them back in the next day.

Work at your own pace as there is plenty of time to complete the course. If you feel you are falling behind then take your workbook home and do as much as you can.

Other pupils can help you [9]

If you feel that you can work with a partner successfully it can be very helpful.

Be prepared to accept help from other pupils as well as the teacher.

Read the problem description and get ideas from a friend at the same stage.

Make sure you understand ...

the problem [14]

Make sure you read over the problem descriptions. If you do not understand, then ask.

Read the task carefully, because it helps to know what you are going to do.

your program [2]

Make sure you understand the program before going on to the computer.

any new techniques [3]

If there is any theory being taught about programming then listen and read carefully as this helps you a lot in your practical work.

Always read new techniques thoroughly, ask for help if you need to because they will turn up again.

And if you are not sure or get stuck ...? [14]

Re-read the question if you don't understand.

Get the teacher to explain anything you're not sure of.

If you are stuck when writing a program, refer back to a similar one, which will guide you.

If your programs don't work look back at past problems which are similar.

Check previous listings to help you.

If you are not absolutely sure, then try it on the computer. This will give you an idea of which part of your program is not quite correct.

Look up "Help Hints". This will give you a hint, and will not tell you the answer.

[If your program doesn't work] get Help Hints to help you, if that fails then you need a teacher to lend a hand ...

When your program doesn't work get a printed listing and check that everything looks "normal".

Retention and revision [11]

Try to memorize all the new techniques you learn so that you don't have to keep backtracking.

Try to solve problems yourself first to help you later, instead of giving up and asking the teacher.

Take your time, because if you rush you will not remember how to do a certain thing. e.g. how to loop may be in more than one problem. If you don't know how to do it first time, it will make it more difficult the second time.

If you are doing an assessment it is a good idea to look back over the topic before beginning.

Do the revision sheets as soon as you have finished the topic so the answers are fresh in your mind.

When doing homework use the summary sheets provided when necessary. Also use these as revision material.

It isn't easy ... [3]

Don't expect it to be easy, it's not all games.

... but you can do it! [3]

New things may seem hard, but they become clear after a few tries.

Go through it one step at a time, don't try to do the whole thing at once.

Design, implementation and testing [6]

Try to follow your top level designs very closely.

Although the Mode 7 design grids are at first pointless, hang on to them as they are helpful as your program nears completion.

It is sometimes helpful if you write out the program first before you key it in. Also make sure you follow the design for the program.

When writing a program use slashes and insert your own comments. This will enable you to know what each part does immediately and so will help you when you have made an error and something has to be changed.

When writing test reports, do not go "over the top". This is time consuming and confusing. Check with your teacher that the situations you have chosen are sufficient.

Always test that a program works perfectly before obtaining a sample run, otherwise you will have to bin a lot of printouts and will get them mixed up.

APPENDIX 5B REVIEW TASK 3 – RESPONSES OF GROUPS AND INDIVIDUALS

Dawn, Bernadette, Gerard

1 *Read over the problem description more than once, and make sure you understand it before you start.*
2 *Organize printed listings, sample runs etc. as you go along and don't let them pile up. This will save time, and will make it a lot easier.*
3 *Look up "Help Hints". This will give you a hint, and will not tell you the answer.*
4 *Take your time, because if you rush you will not remember how to do a certain thing e.g. how to loop may be in more than one problem. If you don't know how to do it first time, it will make it more difficult the second time.*
5 *Get the teacher to explain anything you're not sure of.*
6 *Discuss with your friend.*
7 *If you are not sure if a problem will work, test it out on the computer and you will be informed on your mistakes.*
8 *Remember to save your work.*
9 *Keep your diary up to date so you can remember what you done the last day, and can ask the teacher if you had difficulty with any of the previous work.*
10 *Get your work corrected so you know if you've made any mistakes and you can then go back and correct any errors.*
11 *Do revision questions, in preparation for the exam.*
12 *If you have been off or missed any work, come up at lunchtimes, as it can be easy to fall back on the work.*
13 *Don't change your password too often or you will forget what you have changed it to. Mrs xxx will go mad and not let you on the computer.*
14 *Don't hog the printer or Mrs xxx will ... (see above).*

Paula, Chris, Kirsty

1 *Put the topic and task number, because it is easier to sort them out.*
2 *Read the problem description and get ideas from a friend at the same stage.*
3 *New things may seem hard, but they become clear after a few tries.*
4 *Go through it one step at a time, don't try to do the whole thing at once.*
5 *Always load in the printer driver as soon as you log on. This saves getting annoyed.*
6 *If you are doing an assessment it is a good idea to look back over the topic before beginning.*
7 *Try to solve problems yourself first to help you later, instead of giving up and asking the teacher.*
8 *If you like a particular program, keep a sample run as you have to write about it at the end of the topic [unit]. This helps searching through, trying to find the program later.*
9 *Try to get a chance to use all types of computers.*
10 *Do revision questions straight after a topic because you usually forget things after a while and they help you to remember what you learned.*

Catherine, Kelly

1 *Read over the problem description a few times, then write it down in your own words so you know what to do.*
2 *Organize your printed listings to save time.*
3 *Make sure you understand the program before going on to the computer.*
4 *Do the revision sheets as soon as you have finished the topic so the answers are fresh in your mind.*
5 *Ask for help if you get stuck with a task or don't understand it.*
6 *Use the help hints to see if you can solve the problem yourself.*
7 *Keep all your work together in one folder.*
8 *Remember if you take things home always bring them back in the next day.*
9 *Try to do some work at home to push yourself forward.*
10 *Remember to save your work before leaving the computer.*

Scott, Bryan

1 *Read the task carefully, because it helps to know what you are going to do.*
2 *Plan the steps in your head so that you have a clear idea of what's going on. This will also save time.*
3 *Make sure your plan of action works, write your ideas down.*
4 *Go on the computer and start your task.*
5 *Run your task to see if it works.*
6 *If it doesn't get Help Hints to help you, if that fails then you need a teacher to lend a hand.*
7 *Once it's fixed run it to see if it works.*
8 *Get a printed listing and a sample run.*
9 *Arrange the printouts to save time.*
10 *Save the program!*
11 *Operation Complete!*

Claudia, Kathleen

1 *Make sure you read over the problem descriptions. If you do not understand, then ask.*
2 *If you are not absolutely sure, then try it on the computer. This will give you an idea of which part of your program is not quite correct.*
3 *Make sure you write the topic and task numbers at the top of your printed listings and outputs. And staple them together after every topic. This will be quicker for both you and the teacher when she comes to correct them.*
4 *Take your time to learn where the keys are on the keyboard. This makes your keying in faster for long programs.*
5 *If there is any theory being taught about programming then listen and read carefully as this helps you a lot in your practical work.*
6 *Always save your tasks as the name given in the task sheets. This makes it easier when trying to find programs.*
7 *Do not tell anyone your password. Because some people will not hesitate in going into your user area and changing or deleting programs.*
8 *When doing homework use the summary sheets provided when necessary. Also use these as revision material.*
9 *If you feel that you can work with a partner successfully it can be very helpful.*
10 *It is sometimes helpful if you write out the program first before you key it in. Also make sure you follow the design for the program.*

Anthony, Stephen, Carol-Anne

Anthony:

1 *Keep your folder tidy and keep all printouts in order.*
2 *Write in your diary where you are after each period to save you referring to your book to find out where you are.*
3 *Save your work in case you lose it.*
4 *Don't tell anyone your password.*
5 *Re-read the question if you don't understand.*
6 *Check previous listings.*
7 *Don't expect it to be easy.*
8 *Edit programs by using the copy key.*
9 *Learn to use the keyboard.*
10 *When you have problems ask your teacher for help.*

Stephen:

1 *Write in your dairy at the end of every period to save time and effort finding out where you are in the book.*
2 *Keep your folder tidy and keep all printouts in order so you don't get in a muddle.*
3 *Save everything every so often so that you don't lose any work.*
4 *Don't tell anyone your password in case they do anything shady.*
5 *Re-read the question until you understand it.*
6 *Look back to old programs when you don't understand the work.*
7 *When your program doesn't work get a printed listing and check that everything looks "normal".*
8 *Try to memorize all the new techniques you learn so that you don't have to keep backtracking.*
9 *Don't expect it to be easy, it's not all games.*
10 *Be aware of what the keys can do and get to know the keyboard.*

Carol-Anne:

1 *Keep your folder tidy, it's easier to find things.*
2 *Keep all printouts in order, that way you wont lose them.*
3 *Fill in diary at the end of every period to save time looking up your book to see where you were.*
4 *Save all files as soon as you're finished.*
5 *Don't tell anyone your password, they can change it if you do and then you can't get into your area.*
6 *If you don't understand a question, re-read it until you do.*
7 *Check previous listings to help you.*
8 *If you are having problems, ask the teacher for help.*
9 *Don't expect programming to be easy.*
10 *Save yourself time, use the copy and arrow keys.*

David

1 *Read over the problem description more than once and write it out in your own words, to make sure that you understand the problem before you start.*
2 *Organize your printouts as you go along and don't let them pile up. This will make it easier to refer back to your previous examples and will save you time.*
3 *Don't be afraid to ask the teacher for help if you are unsure about something.*
4 *Be prepared to accept help from other pupils as well as the teacher.*
5 *Try to follow your top level designs very closely.*
6 *Although the Mode 7 design grids are at first pointless, hang on to them as they are helpful as your program nears completion.*
7 *Work at your own pace as there is plenty of time to complete the course.*
8 *If you feel you are falling behind then take your workbook home and do as much as you can.*
9 *If your programs don't work look back at past problems which are similar.*
10 *Write out procedure names to avoid confusion as your program nears completion.*

Caroline

1 *Read over the problem description more than once and write it down in your own words, to make sure that you understand it properly before starting to solve it.*
2 *Keep a note of your progress in your diary. This will save you from wasting time by re-doing things because you forgot you started them.*
3 *Staple listings and runs (for the same program) together. This will help you keep them organized and to find them again when necessary.*
4 *Always save under the name given in the book, so that you will not get mixed up with different titles.*
5 *Always keep a sensible password. This will prevent files being changed by another person gaining access to your area.*
6 *If you are stuck when writing a program, refer back to a similar one, which will guide you.*
7 *Always read new techniques thoroughly ask for help if you need to because they will turn up again.*
8 *When writing a program use slashes and insert your own comments. This will enable you to know what each part does immediately and so will help you when you have made an error and something has to be changed.*
9 *When writing test reports, do not go "over the top". This is time consuming and confusing. Check with your teacher that the situations you have chosen are sufficient.*
10 *Always test that a program works perfectly before obtaining a sample run, otherwise you will have to bin a lot of printouts and will get them mixed up.*