THE MOTIVATION OF STUDENTS OF PROGRAMMING

a thesis submitted to

The University of Kent at Canterbury

in the subject of computer science

for the degree

of master of science.

By

Tony Jenkins

September 2001

Contents

List of Tables

List of Figures

Abstract

Teaching (or, perhaps more accurately, *learning*) computer programming is a problem. Many students struggle when they first encounter a programming course and many graduate with little confidence in their programming ability. Since programming is a fundamental part of the discipline of computing and since skilled programmers are much in demand in the IT industry this represents a singularly depressing state of affairs for computing educators.

This thesis presents an investigation into a key psychological factor in programming students – their *motivation*. The students at two UK universities were surveyed at three important points in their programming courses during the 2000/01 session. The surveys focused on the reasons why these students had chosen to study an IT degree and on the development of their feelings about their degree course (and about the programming in particular) through their first year. This research was complemented by a further study of a smaller group of students who were followed in detail on a weekly basis through their programming course. Their aspirations, struggles, and experiences were recorded.

Students of any subject will not learn if they are not motivated. This thesis highlights a number of issues surrounding the motivation of programming students. They appear to value the outcome of their course throughout but they often lose motivation because they cease to believe that they will succeed. If they lose motivation they will not learn.

A teacher of programming has a responsibility to pass on that skill to the students. But there is a more important responsibility. A teacher of programming must above all make sure that the students are motivated. The students must *value* the learning outcomes and they must also *expect* success.

Teaching programming is only a small part of the story. Motivating students as they learn programming is a bigger, much more important, part.

Acknowledgments

There are many people I need to thank for their help during this work and so I make no apology for the length of this section.

First, thanks go to Janet Carter and Ken Brodlie. Without you I wouldn't have finished this. Thanks. Thanks also to Janet for lending me the corner of her office when I was in Kent and thanks to Janet and Mark Wheadon for lending me their spare room and making sure I was fed and 'watered'. A special nod goes to Mark's quite extraordinary garlic bread.

My supervisors, Ursula Fuller and Stefan Kahrs, provided many useful ideas, much advice, and (successfully) challenged my initial ideas. Thanks also to others who helped at Kent: Ian Utting kindly distributed the questionnaires and sent them back to me with quite amazing speed, Sally Fincher and John Slater started the whole thing off, and Simon Thompson provided encouragement and travel money. Roger Boyle and David Hogg at Leeds also prodded things along. Karim Djemame, John Davy, and John Hodrien tolerated interruptions to their lectures in Leeds and helped with collecting the questionnaires.

Thanks also to the proofreading team – in vague order of pedantry, Stan Jenkins, Simon Myers, Janet Carter (again), Dave Ansley, and Rik Wade. Any errors that remain have presumably been inserted by my enemies.

Jemma Pauley kindly did most of the things that I should have been doing this summer. And she didn't get too upset when I got grumpy. Ta – you're a star.

This work would not have been possible without Stacey Lewis, Clare Viner, Ian Fletcher, Jill Shutt, Ozzie Thomas, Rachel Wottge, Simon Cubitt, John Kelly, Dan Sherburn, Richard Whitefoot, Lucy McGregor, Sophie Cottam, Liran Kessler, Neil Meikle, Roni Chakrabarti, Caroline Wells, Phil Mellor, Shane Seddon, Adam Rideout, Herry Basuki, Leanne Freedland, Simon Perry, and (last and far from least) the twins – Louise Manley and Tessa Price. They may recognise themselves in chapter 5.[1] Thank you all – I don't deserve you.

Much the same goes for all the students at Leeds and Kent who kindly completed the questionnaires. Many thanks for your time.

Thanks also to Dave Ansley and John Wallace for selflessly agreeing to consume large amounts of fine ales and beers so that I didn't spend too much time worrying about the last full stop. 251 – not bad. Dave also provided the document solution.

Much assistance with the mysteries and black arts of LaTeX was received from Henry Bell, Nick Efford, Sarah Fores, Chris Goodyer, Chris Needham, David Pashley, Rick Peterson, Qef Richards, and Mark Walkley. Thanks. LaTeX is indeed, as Rick noted, "a worthy adversary". Chris, Chris, and Mark also sorted me out with the intricacies of `gnuplot`. Sarah frequently lent me her ruler.

Carlos Fandango, Steve Harris, and William Towle helped with a tricky reference [125]. Thanks, chaps.

Finally, in no particular order, respectful thanks to John Pemberton, Joshua Tetley, the person-who-invented-coffee, Dylan Thomas, Ron Ridout, the Leeds and Liverpool canal, S. A. Brain, Jack, Scott Adams, whoever it is that pays for email, and everyone else I've forgotten.

If I'm allowed a dedication this is for Dominique and Stacey. Diolch yn fawr i chi'ch dwy.

Tony Jenkins
Leeds, September 2001

[1] There is actually a trick to working out who's who (I would have gone mad without one) but it's a secret.

More Acknowledgments

I must record thanks to my examiners, Pete Thomas and David Barnes, for their constructive comments on the first version of this thesis. Thanks are also due to David for his prompt help and advice while I was carrying out the required corrections.

I now realise that I forgot to acknowledge the Staff and Departmental Development Unit at the University of Leeds for partly funding this work. Thanks and apologies.

Thanks are now due to former feedback-meister Matthew Hubbard for his in-depth knowledge of the UNIQoLL project.

Mark Conmy provided some singularly arcane Perl to deal with the thorny problem of the 'Oxford commas' in the references. It seems a shame that I eventually put them back in.

Mark Wiggins sorted me out with some useful information about juggling.

Thanks once again to the proofreading team – (once more in order of pedantry) Heather Gulliver, Stan Jenkins, Simon Myers, and Dave Ansley. Special thanks to Dad, Smylers, and Dave, who all did the job for the second time. I blame the errors that remain on the printer elves. Dave also once again supplied a document solution.

I realise that I still have Sarah's ruler. I must return it.

I never did find out whether I was allowed a dedication but just in case this time it's for Heather. Ta. You is a star.

Tony Jenkins
Leeds, July 2002

Publications

While none of this thesis has been published elsewhere in its present form, work based on that presented here (and some preliminary findings) has been published, as follows:

- A preliminary version of Section 4.2 formed the basis of *The Motivation of Students of Programming*, which is published in the Proceedings of ITiCSE 2001, Canterbury, UK, June 2001, pages 53–56. This is included in the references as [74]. This paper is © ACM.
- Some of the observations about motivation in Chapter 3 were used in a poster – *Motivation = Value × Expectancy* – the abstract for which is published in the Proceedings of ITiCSE 2001, Canterbury, UK, June 2001, page 174.

More Publications

Since the first submission of this thesis the work presented here has formed the basis of a few conference papers and other presentations as follows:

- The overall experience led to the writing of *Teaching programming – A Journey from Teacher to Motivator*. This is published in the Proceedings of the 2nd Annual Conference of the LTSN Centre for Information and Computer Sciences, September 2001, pages 65–71. The text of this paper is also available on-line at `http://www.ics.ltsn.ac.uk/pub/conf2001/`. This paper is © LTSN-ICS.

- This work, especially the reflections on Dijkstra's classic paper, also inspired *On the Cruelty of Really Teaching Programming*, which was presented at the 2nd LTSN-ICS One Day Conference on the Teaching of Programming at the University of Wolverhampton in March 2002. Enthusiasts can find the slides from this talk at `http://www.ics.ltsn.ac.uk/pub/prog2/`.

- The section on the difficulties inherent in learning to program is the basis of *On the Difficulty of Learning to Program*, which will be presented at the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, at Loughborough in August 2002. This paper will be © LTSN-ICS.

Trademarks

The following trademarks are used in this thesis and are acknowledged. It is possible that other trademarks, of which the author is not currently aware, may have been unintentionally missed.

- Acorn is a trademark of Element 14 Ltd.
- CodeWarrior is as trademark of Metrowerks Inc.
- Delphi and Turbo Pascal are trademarks of Borland International.
- Frisbee is a trademark of WHAM-O Inc.
- Icky Poo is a trademark of Klutz.
- Java and JavaScript are trademarks of Sun Microsystems Inc.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft, Microsoft Windows, Microsoft Windows NT, and Microsoft Visual Basic are trademarks of Microsoft Corporation.
- Unix was a trademark of Unix System Laboratories Inc. and is now registered to the X/Open Consortium.

These trademarks are denoted ® or ™ as appropriate the first time they appear in the text.

Chapter 1

Introduction

First Voice: To begin at the beginning . . .

Programming is a subject that lies at the very heart of computing. A computer is quite useless unless it is running a program and an understanding of how such programs are written is a key part of the development of any computing student. As such it is not surprising that one of the fundamental courses in any degree-level computing programme is one intended to teach the students to program.

In the UK today students approach the study of computing in higher education in increasing numbers from an increasingly wide variety of backgrounds. They range from complete novices (few computing degree courses require any formal academic qualification in the area) to those with substantial experience of working in the IT industry. Even among those who have studied computing at 'A' level or equivalent there is rarely any extensive experience of programming. The number of students taking computing degrees is increasing and there is currently no sign of an end to this trend. As the numbers increase further so will the diversity.

Given its importance it is disappointing to realise that the teaching of programming (perhaps, more accurately, the *learning* of programming) is a perennial problem.

Experienced teachers are all too familiar with the struggles of new students as they attempt to come to terms with this most basic area of expertise. Many teachers will have seen students choose course options (or even change degree programme) in order to avoid more programming and most will have been faced with final-year students approaching a project or dissertation determined to avoid undertaking any programming at whatever cost.

The overall objective of this study is to learn more about the experience of students learning to program in today's higher education system. Much has been written about the best language, paradigm, or environment for teaching programming but relatively little has been done so far to form a solid understanding of the problems, pressures, and experiences faced by the students as they attempt to learn to program. It is odd that so many innovations in teaching programming seem to have taken place with little or no reference to the group that the change arguably affects the most – the students. Few proposals advocating a change in language, paradigm, or environment (and there are many) offer any compelling evidence that the argued-for change would represent an educational advantage for the students.[1]

The key issue in this study is *motivation*. In essence, students will not learn if they are not motivated and they will not be motivated unless they believe that they will succeed. Little is understood about the reasons why students take programming courses (or indeed computing courses in general) and there must be a suspicion that the institutions teaching them have failed to appreciate that the motivation of the students has changed (or have perhaps failed to understand it in the first place). Perhaps it is no longer safe to assume that students enrolled on a computing course have any particular interest in computing, or that students taking a programming course have any particular interest in learning to program. It may well be that their main reason (their motivation) for choosing their course is that a degree will give

[1] A similar observation has been made in the field of mathematics education [138] – "Study after study in the math-ed literature produced 'promising' results, where teacher and students alike were happy with the instructions, but where there was little evidence, if any at all, of improved problem-solving performance.".

them access to a lucrative career and that they see the programming course as simply one among the many courses that they are required to pass. A teacher faced with such a class and failing to motivate the students to value the learning is unlikely to succeed in teaching them to program.

Very little is known in general terms about what motivates students to choose to start a degree course or to choose a particular subject [72]. A recent study in the School of Computing at the University of Leeds [28] appears to confirm the suspicion that a (perhaps the) prime motivator for students taking a computing degree is the prospect of a highly paid job as a programmer. It certainly appears to be the case that there are differences in the motivations for choosing different subjects [72] and in the current social and economic climate it is not an unreasonable hypothesis that the prospect of a financially rewarding career is a strong motivator for any student choosing to study computing.

Learning to program can often appear, from the outside at least, to be an extremely stressful experience. At times it seems to take over the lives of the students as they lurch unhappily from one summative assessment to the next. At present this view is anecdotal, and indeed largely personal, albeit based on experience over several years and confirmed by many teachers at many institutions in many countries [102]. In order to teach programming better teachers must understand more about the experience of their learners and more about what motivates these learners. While there is plentiful research on students' 'learning styles', and the effects of the academic environment on these styles, there is much less written about what actually happens when students learn.

The work presented here takes two complementary views of the students' learning. The first is a wide view (the experiences of an entire cohort of students) and this is then complemented by a more detailed view (the weekly experiences and feelings of a much smaller group). Specifically, this study presents the results of an investigation into the experiences and attitudes of students studying the introductory computer programming courses at the University of Kent at Canterbury and at the University

of Leeds. These are two 'traditional' UK universities, recruiting cohorts of students of a similar academic standard from roughly the same pool [24]. Previous statistical work [16] has shown that the two cohorts can reasonably form the basis of such studies. The wider view is provided by the students at both institutions who were followed throughout their first programming course by means of questionnaires completed at three key points. To complement this part of study the more detailed view is provided by a smaller group of students at the University of Leeds who were followed on a weekly basis through the first semester of their programming course.

The objectives of this study are, then:

- To understand the *experience* of learning to program in today's higher education system in the UK.
- To understand how the students' *motivation* for and *attitudes* towards their programming course alter as the course proceeds.
- To understand the reasons why students choose to take programming courses.

These objectives clearly require a definition of programming. This is a difficult thing to find since it is unlikely that any two educators or programmers would be able to agree on any precise and unambiguous definition. As a possible starting point the *Oxford English Dictionary* [119] provides:

> The operation of programming a computer; the writing or preparation of programs.

But this seems very narrow. *Merriam-Webster* [108] is scarcely any better, with the only promising definitions on offer being:

> 2 : to work out a sequence of operations to be performed by (a mechanism)
> : provide with a program;
> 3 a : to insert a program for (a particular action) into or as if into a mechanism; b : to control by or as if by a program.

Programming is indeed a difficult thing to define! Indeed, one of the examples of usage given in the OED includes "The *final* stage of programming must therefore consist in the translation of the flow diagram into actual coded orders which the machine can understand" (italics added for emphasis) – a clear implication that there are other stages involved in this mysterious process.

For the purposes of this study it is hoped that a none-too-contentious definition of programming can be adopted. Programming is therefore defined here as "the process of taking a problem specification written in plain language, understanding it, devising a solution, and then converting the solution into a correct computer program (usually expressed in some special-purpose programming language)". This process encompasses a number of identifiable tasks and phases (which will be discussed in more detail later on) but in overview this is the definition that applies in this work.

Programming is thus a set of processes that together form an identifiable skill. It can be described as 'doing-centred' [136]. Acquiring this skill requires the development of a range of abilities of different types and requires a great deal of practice. Applying such a skill usually becomes much easier and natural after significant practice and experience.

The acquisition of this skill requires that the learners do several things. They must master the syntax of a particular programming language and they must understand the semantics conveyed by the syntax. They must be able to understand and analyse a problem, devise a solution, and then express their solution in the programming language. They must be able to demonstrate that their solution is correct and that their implementation of the solution is correct. Experience shows time and again that these are difficult things to achieve.

This is a study about motivation. The motivation for carrying out this study comes from the experience of several years devoted to teaching the various incarnations of the introductory programming courses in the School of Computing[2] at the University

[2] Pedantically, for only two years in the School of Computing but for several more in the School of Computer Studies.

of Leeds. I[3] have spent many hours presenting an introductory programming course in one of the most widely respected computing departments in the UK. I came to the task by accident when a colleague was unavailable for one year and I was asked to fill in. My teaching background was limited and what I had was concentrated in the area of databases and fourth generation languages. Still, it was explained to me, teaching such basic material as programming is not a problem.

For my first delivery of the module the language I taught[4] was Pascal in a Unix® environment. I presented the lectures following the materials bequeathed to me by my predecessor and I dealt with a steady flow of students knocking on my door seeking assistance. Coursework was issued and marked, and I enthusiastically checked through it for evidence of plagiarism. It was with genuine surprise that I discovered at the end of my first delivery of the module that many of my students simply could not write even the simplest of programs. They had completed my assessments well enough and had secured sufficient marks to pass (which was in itself worrying) but they still could not program in any meaningful sense. While they might have passed the assessment to an acceptable extent I would certainly not have employed them as programmers and I could not see that any self-respecting employer would. What was I doing wrong?

Since this first outing I have spent many hours thinking and writing about the ways in which I teach programming and have, I think, achieved some modest success in devising effective ways to learn and teach. The work described here is largely the result of the opportunity presented by the chance to take a year's break from teaching programming and the resulting chance to see the course once more from the outside. I have now told several groups of students over several years that teaching programming is, to me, "the best job in the world". This is something that I still believe. I doubt that they believe me but I have missed doing it for a year. Being able to write this

[3] To avoid many awkward sentences in what follows the author hopes that he may may be permitted to use the first person briefly. He will not use it again until the final chapter.

[4] I confess. I taught a language. I did not teach programming. Only now do I appreciate the difference.

thesis has helped and I hope that this work has enabled me to become a better teacher of programming. I also hope that others may find something in it that might improve their own teaching.

In the thesis that follows chapter 2 presents some of the relevant background issues and the educational theory. These are made specific to the learning of programming in chapter 3 which also considers the peculiar features of programming that make it so difficult to learn. The results of the questionnaires presented to the whole class are described and discussed in chapter 4 and the experiences of the small group of individuals is the basis of chapter 5. Finally chapter 6 reflects on what the study has shown about the learning (and teaching) of programming.

A few specific details of my involvement with the programming course at Leeds follow, and will help with the understanding of this work as a whole (particularly chapters 4 and 5). I have been involved in teaching the course since the 1994/95 session. The session in which this work was carried out (1999/2000) was the first for several years when I had had no direct involvement. During this year I helped out (in a strictly unofficial capacity) with some of the practical sessions but was not directly involved with any of the more formal teaching. I was, therefore, only slightly better known to the students at Leeds surveyed in this work than I was to those at Kent (although I was presenting a different module to some of the Leeds students).

I had taught the course during 1998/99 and so was able to select the students whose experiences form the basis of the first part of chapter 5 from my recollections of the course. They were chosen as names I remembered or recognised from the class lists and thus represent a true range of experience.

The individual novice programmers in chapter 5 formed my first year tutorial groups in the 1999/2000 session (there were three groups). I had no direct responsibility for teaching them programming but they did seem to call by with their C++ problems from time to time. They were effectively selected from the whole class at random.

The students whose experiences formed the basis for this work were all students of programming. During their time on a programming course they faced a range of challenges, not all of which were directly related to their course or to programming in particular. Other students on other courses were also trying to come to terms with the usual problems and pressures of student life and of the first year of a university course. They too were very probably presented with difficulties quite peculiar to their own chosen disciplines.

It is not claimed that all the experiences of the students recorded here are unique to students of programming. Far from it. However, the subjects of this study are programming students and the focus is on all their experiences.

Chapter 2

Background

First Voice: Listen. Time Passes. Listen . . .

A single monolithic model or view of learning is an unattainable objective and so this study will need to draw on several areas of background research. To this end this chapter begins by describing some of the relevant theories and ideas from motivational theory. It then introduces the various aspects of the theory of learning, which will be referred to in later chapters. The chapter concludes with a consideration of some of the more recent changes in higher education in the UK and the impact that these have had on the academic and institutional climate in which students now learn to program.

A common theme in these sections is *assessment*. Assessment provides the basis of the final measure of a student's ability or learning and is, as such, extremely important. It has an impact on the way in which students study and learn, on the students' motivation, and on the crucial relationship between the teacher and the students. The students are normally more than a little interested in the results of the final assessment (the grade that they receive at the end of the module). In these sections 'final assessment' refers to just this and nothing more. This result

may well be the cumulative result of a number of assessment methods, including continuous assessment and perhaps formal examination. The precise method will differ for different institutions and different subjects. The importance of assessment as a background theme in this chapter cannot be overstated.

2.1 Motivation

The motivation (or otherwise) of students is the key issue in this study. There is an intentional double meaning here. The two issues are *motivation* as an attribute of students (that which makes them want to succeed or makes them work and learn) and the teacher's crucial rôle in *motivating* a class of students. To succeed in any academic task students must be motivated and they must want to succeed. Biggs [14] neatly defines a teacher's motivational rôle as "getting students to agree that appropriate task engagement is a good idea". If the teacher can devise suitable teaching (or learning) tasks, and can then persuade the students to engage in them, the students will learn (or will be unable to avoid learning).

While it may be reasonable to assume that all the students studying a course are motivated to succeed[1] (at the outset at least), it is unreasonable to assume that they are all motivated for the same reasons. Certainly it is completely wrong to assume that they are interested in learning the content of the course for its own sake! An understanding of what motivates students is essential if they are to be taught effectively and if they are to learn [29].

Unfortunately, motivation is an inherently abstract concept that is difficult to measure or identify in any meaningful way [4]. It is possible to observe behaviour and from that to infer an individual's likely motivation but it is never possible for an observer to be certain. Motivation is a deeply personal concept [49]. If subjects are questioned directly about their motivation the questioner can never be totally certain that the

[1] This is probably debatable (but a student who had no motivation to succeed would almost certainly soon drop out).

subjects are telling the truth and can never be completely sure that the inferences made on the basis of the questioning are correct. Even if the answers given are completely honest there still remains the need for some speculation on the part of the questioner and therefore some uncertainty always remains.

It is more straightforward to provide a taxonomy of possible motivations for some person undertaking some task. Of particular interest for this study are, of course, the general categories of motivation that can be observed in an attempt to describe why students might value learning (and so engage in those tasks and activities intended to make them learn).

2.1.1 The Value of Learning

As a starting point, Fallows and Ahmet [53] propose a rather informal list of reasons (presented in no particular order of importance) why a student might value learning:

- the learner's desire to please the teacher.
- perceived need [to understand] the material presented.
- each learner's degree of interest in the subject material.
- the personal philosophical values and beliefs of the learner.
- the learner's attitudes to the materials being delivered.
- the academic and career aspirations of the learner.
- incentives and rewards which are expected to accrue from the learning.

Clearly some of these factors will be stronger than others (and some are unlikely in a higher education context where it is hard to imagine many students setting out mainly to please their lecturer). The degree to which each factor will have an influence over a particular student's motivation will be different in each case. It is also likely that for some students some factors will be completely absent. The key task for the teacher

faced with this complex situation is to inspire the students by maximising the positive effect of each of these factors for each individual.

Entwistle [50] takes a more general view and so describes three rather more generic types of motivation (or reasons for valuing learning):

- *extrinsic* – the desire to complete the course for some expected reward.
- *intrinsic* – deriving from interest in the subject.
- *achievement* – based on a desire to 'do well' and (sometimes) perform better than peers.

It is clear that students motivated primarily by any one of these three types of factor will have very different approaches to their studies. A student motivated primarily by extrinsic factors, for example, would probably do very little for which there was no summative assessment credit. Students with intrinsic motivation could be expected to read around the subject and form their own views on the material that they were learning. If achievement is the prime motivator the students will adopt whatever strategy they think will gain the best rewards, or the approach that will allow them to perform 'best' (as measured in the final summative assessment). All these students will engage and will probably learn. It is their reason for engaging (and learning) that is different.

These three examples represent stereotypes. Each of these three factors will exercise an influence over each learner to some extent, although for most learners one factor will be of most importance [53]. In computing in particular it is hard to see that extrinsic motivation would be totally absent for any student but until some evidence emerges one way or the other this must remain a distinct possibility.

Entwistle's list omits the "desire to please to teacher" proposed by Fallows and Ahmet. While it is unlikely that students will be motivated first and foremost to please their lecturer, as suggested by Fallows and Ahmet, it seems quite likely that they would be motivated to please some other party whose views are important to them or to

whom they feel in some sense accountable (their family or sponsor are the obvious possibilities). Biggs calls this *social* motivation [14].

Social motivation might also include to some extent the fear of failure (a very strong motivator for some (especially women [60])). In any case, some students may be highly motivated simply because they do not want to fail and so be a disappointment to someone whose opinions they value. They may even be responding to threats of dire consequences should they fail [113]. Biggs's social motivation, as a factor that makes a student value a learning experience, can therefore be extended to include the fear of failure and its possible consequences. This also adds to the more specific list of Fallows and Ahmet and to Entwistle's more general offering.

There remains the final (rather unhappy) possibility that there will be students who are completely lacking in motivation. This might be for a variety of reasons. They may just be disenchanted with education or they may believe that they are taking the wrong course. Hopefully there are very few such students but it is a possibility. To account for this eventuality a fifth category, *null*, will also be used in this study to cover those students who have no particular motivation at all.

This gives the following five categories of motivation:

- *extrinsic* – the primary motivation is the career and associated rewards that will follow from the successful completion of the course.
- *intrinsic* – the primary motivation is a deep interest in computing (or specifically programming) for its own sake.
- *achievement* – the primary motivation is to perform well for personal satisfaction (this satisfaction may also derive from out-performing peers).
- *social* – the primary motivation is a desire to please some third party whose opinion is valued.
- *null* – there is no particular motivation (such a negative view might at best be characterised by the statement "I just want to pass").

It is important to emphasise that programming students falling into the first four categories all *value* the learning to be gained from a programming course. Even students in the final category attach some sort of value to it but it is reasonable to suppose that this is in some sense less (in both magnitude and desirability) than the students in the other four. This study will argue that in many ways it matters not why students value a learning opportunity (and its outcomes) just so long as they do.

This list is somewhat simplistic but this is only to be expected with any taxonomy of so abstract a concept as motivation. It must be emphasised that it is likely that students will derive their motivation from more than one of these categories. A particular example might be a deep interest in a subject (intrinsic motivation) developing as a result of expected eventual rewards (extrinsic motivation). Nevertheless, the present taxonomy will serve to identify a student's dominant motivation.

2.1.2 Expectancy and Value

These views of motivation attempt to define how or why a learner *values* a learning opportunity. This is only one part of the story. For students to be properly motivated they must also be able to *expect* success. There is no point in students being highly motivated to succeed in something that they also view as impossible. A task may well be viewed as very difficult (indeed, this could be an additional motivator for some who may relish a challenge) but a student must always regard the learning task as possible.

The recognition of the existence of these two interacting factors leads to the popular expectancy-value theory of motivation (for example [81]). This theory provides a view of the extent of motivation as a function of two connected factors:

- the *value* that the learners attach to the outcome.
- the extent to which they *expect* to be successful in the final assessment.

These two crucial components[2] are said to multiply rather than add:

$$motivation = expectancy \times value$$

They must both be positive (and non-zero) for there to be true positive and effective motivation. It follows that if either of the factors falls to zero so does the product. Students who do not expect to succeed will not be motivated no matter how much they value the potential outcome. Similarly, students who attach no value to the outcome will not be motivated no matter how much they expect that they might succeed.

If it is safe to assume that all students who start a course of study have some sort of motivation that makes them value the outcome of their learning the question of expectancy then becomes critical. Students must believe that it is possible to achieve a reasonable result (with 'reasonable' defined as that standard which meets their own expectations or aspirations). Their initial views on their likely chances of achieving this will come from the teacher – academic success may not be easy to achieve but it should not be impossible. This message must be conveyed to the students but with care. This also has clear implications for the way in which assessment, and feedback on assessment (which contains the important message about a student's likely success), is carried out. A teacher has a crucial rôle to play in ensuring that the students expect success.

2.1.3 Factors in Motivation

If motivation is seen as a multiplicative function of two factors two questions arise:

- What affects the magnitude of the factors?
- What influence can a teacher hope to exercise over the factors?

[2] A few writers include a third factor in this equation, *affect*. This covers a student's emotional response to the learning experience. This factor is more usually omitted, as in this work.

An attempt to answer these questions can be based on the existing wide body of literature on motivational factors in employment. This can reasonably be applied to student learning since, effectively, a student's employment is learning [49].

In the workplace an employee's level of motivation is influenced by many factors. Only some of these are under the employee's direct control. Working conditions, for example, can have an impact on motivation but are often something that the worker cannot directly change. Other factors – such as 'sense of achievement' – are internal to the worker and as such cannot be directly affected by the employer.

Applying this observation to learning, there are factors that can be affected by the teacher ('the way the students are taught'). Equally, many factors cannot be explicitly addressed by the teacher, such as the students' level of interest in the material being taught. At the same time a teacher can still attempt to exercise some influence over the internal factors. Teachers who appear enthusiastic, and who appear to enjoy their subject, can hope to pass some of their enthusiasm and enjoyment on to students. A less enthusiastic teacher is unlikely to have the same inspirational effect.

In the 1950s Herzberg [66] argued that the factors affecting motivation fall into two distinct categories. The first of these, which he called *motivator* factors, brings great satisfaction, while the second, *hygiene* factors, can lead to dissatisfaction. He argued that each set of factors functions over only half of the motivational scale (this has since been disputed [49]). The absence of motivator factors, for example, would lead to a neutral state of motivation rather than an unmotivated state. In the case of students the most significant hygiene (negative) factor is generally held to be a lack of preparation for summative assessment (and is therefore closely linked with the *expectancy* part of the motivation equation).

When considering the factors that have an influence on motivation (and also when attempting to identify those that a teacher or manager can hope to influence) a popular view in management science is Maslow's hierarchy of needs [99]. Maslow holds that individuals will not be interested in satisfying their higher-level needs (such as achievement, recognition, and advancement) until their lower-level needs (for example food and warmth) have been met.

Elton [49] applies this hierarchy to the case of students who "cannot be expected to be interested in learning for learning's sake until they are satisfied that their needs to learn in order to pass the examination have been met". It follows from this that one of a teacher's key responsibilities (perhaps *the* key responsibility) must be to ensure that students believe (expect) that they will pass the course's final assessment. This is something that seems to be more an aspect of a tutor's pastoral support rôle rather than of a teacher's instructional rôle.[3] Once this basic need has been met the teacher can move on to attempt to stimulate higher-level factors. This involves inspiring the students with a genuine interest in the subject. They will hopefully then engage more, but only after their most basic need has been satisfied.

Of course, many other factors will have an impact on a student's motivation. The list of possibilities is practically endless – personal life, financial pressures, family problems – and a teacher can probably hope to address only the academic side to any significant extent. It is crucial to ensure that the students *value* the learning and *expect* success in the assessment. The teacher must, therefore, attempt to influence the behaviour of the student and thus exercise some sort of control.

2.1.4 Locus of Control and Personal Causation

Control in an academic context is crucial. In this setting it is the students' basic responsibility to demonstrate to the teacher that they should be awarded a pass. Correspondingly, it is the teacher's responsibility to ensure that the students achieving a pass do indeed merit it. This raises the issue of where the *control* in this somewhat delicate relationship lies. The idea of *locus of control* [90] addresses this. Loosely, individuals may be *internally* oriented and view success as a direct result of personal efforts or they may be *externally* oriented and view success as being in the gift of some powerful individual, available on a whim, and quite independent of their own

[3] There is the possibility that a student is in a position where there is simply no possibility of success because of previous performance. In this case a teacher should surely not lie. The key is to make sure that the student never reaches this state (which should be possible if it is accepted that all students are capable of passing at the start of the course).

efforts. Thus, internally oriented students will believe that they will succeed if they work hard whereas externally oriented students will see success as a gift bestowed randomly by the teacher.

This idea is closely connected to the concept of *personal causation* [39]. In this view individuals are either identified as *pawns* (whose behaviour and objectives are determined by others) or *origins* (who are more in control of their actions).[4] The feeling of being a pawn can lead to rebellion [49] and eventual drop-out or failure.

For effective learning, students must feel in control or else they will quickly become disillusioned. They must feel as if the locus of control lies firmly with them and not the teacher, and also that their success (or otherwise) depends on their own efforts rather than on the perhaps arbitrary decisions of the teacher. In personal causation terms students must be origins in that they must take responsibility for their own learning. This necessity (or expectation) often comes as a shock to students used to the school environment where they are much more like pawns. The teacher has an important rôle here. The students must be persuaded to take control and accept responsibility for their own learning. Some teachers may also find this difficult as they may not be prepared to relinquish control (or even power [48]).

In a teaching relationship the teachers clearly have some power over the students, essentially because they control the students' assessment. Internal orientation (or origin behaviour) is clearly preferable for a student from a learning perspective. A teacher's challenge becomes to ensure that the students feel in control of their own destiny. A powerful and aloof teacher is unlikely to be able to achieve this easily.

2.1.5 Learned Helplessness

Seligman [124] has identified the concept of *learned helplessness*, which is connected to pawn behaviour. This term describes a feeling that develops when an individual

[4] A similar concept to the pawn is sometimes expressed as *amotivation* [40]. This is a state where individuals can see no link between their actions and the outcomes that result from them.

is motivated to succeed in an activity (*value*) but finds it totally impossible to do so (*expectancy*). Keller [81] gives the example of a child studying algebra who for whatever reason misses some vital fundamental concept. The child wants to pass in algebra, and cannot avoid attending classes, but finds it quite impossible to succeed without some additional information (information that the child may well not even realise is missing). This leads to the conviction that the child "just can't do algebra".

Keller's example will strike a chord with any teacher who has been faced with a final-year student approaching a final-year dissertation determined to avoid programming because of a firm conviction that they "just can't do it". Programming is based on a series of fundamental concepts. It is all too easy to see how it would be quite impossible for a student to write a program requiring functions with parameters without a knowledge of variables and data types, or a program including conditional statements with no concept of even the simplest Boolean expression. The learned helplessness response provides a convenient explanation for a student's failure to succeed. It is for many an attractive and reassuring explanation. They may have failed in something, but they have failed in something that was quite impossible for them to pass.

The learned helplessness response is, like motivation, a deeply personal thing. As with motivation it is possible for an observer to suspect that a subject is experiencing learned helplessness, but it is never possible to be sure. Moreover, subjects displaying this type of response will be unaware that this is the case. The problem is simply and genuinely that they "just can't do it". It is only when the subjects are closely questioned (and perhaps even receive some form of counselling) that the observer's suspicions can start to approach a certainty.

Sometimes the student will have missed the basic material through some omission or unexcused absence. In many cases, though, there will have been a good reason such as illness. The progress of some topics in mathematics and computing can often be seen as relentless. Concept builds upon concept as the teacher heads inexorably to the end of the syllabus. In this situation it is not surprising that some students feel

lost and helpless. Learned helplessness is a difficult condition to reverse, but it is not impossible to do so. In attempting to reverse it the teacher's crucial rôle is that of a motivator, providing comfort, reassurance, and encouragement [1].

2.2 Experiential Learning and Constructivism

It is widely argued that learning comes about through a complex process based around *experience* [43] and *reflection* [139]. The prevailing view of this process is summarised by Kolb's Learning Cycle Model [86] (figure 1). In this model a student undergoes some learning experience, reflects on it, forms abstractions from it, and is then able to construct and carry out experiments. These experiments in turn produce more experiences which can be the basis for further reflection and abstraction.[5]

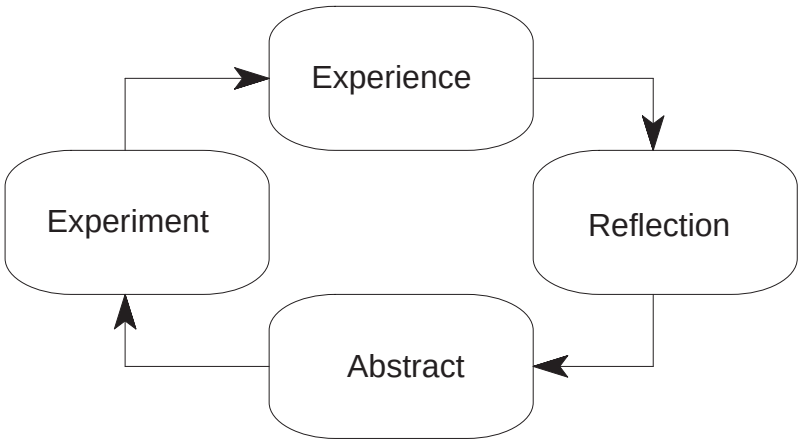


Figure 1: Kolb's Learning Cycle

Other work by Kolb [87] adds four distinct *learning styles* to the cycle of figure 1. One is placed in each quadrant, as shown in figure 2 on the next page. The position of each style in the cycle illustrates the learning strengths of that style. A *diverger* excels in the process of reflecting on an experience, an *assimilator* has a strength in forming

[5] Kolb's model has been widely applied and cited in similar studies to this (for example [21], [27], [69], [73], and [91]).

abstractions from this reflection, a *converger* can develop practical experiments from these abstractions, and an *accommodator* is able to adapt existing knowledge on the basis of an experiment.

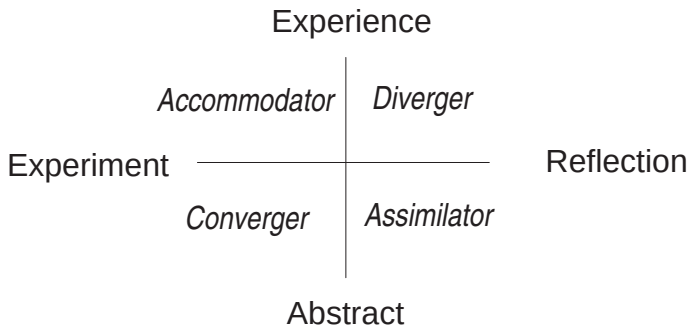


Figure 2: Kolb's Learning Styles

Kolb also associates personalities with each of these rôles. A diverger is a 'people person', an assimilator likes to work with detailed and logical information, a converger does not enjoy working with people (and prefers situations where there is just one answer), and an accommodator is able to adapt quickly to new information.

A complementary view of education (and one that has clear parallels with Kolb's cycle) is *constructivism*. According to this view each learner's knowledge is built up (constructed) as a result of experiences and reflection. Lectures and textbooks have only a passive rôle in this – the process of learning is very much a personal one. It follows that each learner's epistemological view is personal and, as such, is unique to each learner. A teacher's task in a constructivist setting is to facilitate the learner's building of this view. In Kolb's terms this means that the teacher guides the students as they progress round the cycle. This view of the teaching process is similar to Laurillard's model of teaching as conversation [89] or indeed the classic Socratic approach of teaching by participation in debate.

Ideal students would pass through the phases of Kolb's cycle as a natural part of every learning experience. Their rôle in this is what Schön has called a reflective practitioner [139]. The rôle of an ideal teacher, teaching ideal students, is simply to guide them through this process. The task of teaching is to guide the student through the four learning styles in Kolb's model.

2.3 Learning Styles

Kolb's cycle highlights the fact that different students learn best in different ways (or at least that different people will excel in different quadrants of Kolb's cycle). Some can learn effectively from reading books while others prefer to learn by discussing material with other learners. Some learning tasks may be best approached in a particular manner. (The alphabet, for example, is something that can only be learned by rote and memorised.) More complex learning requires a more complex process, the development of an *understanding*. For example in history it is possible to memorise the dates of the events leading to some battle but it is far more complex to develop an understanding of the causes of the battle. The transition to more complex learning is a process which will develop in any learner over a period of time (and will develop over a student's course, becoming more specific to the chosen area of study [117]). In the early stages of any educational endeavour simple learning by rote is likely to be considered acceptable and a necessary precursor to more sophisticated learning.

The manner in which material is conveyed or taught can have a strong influence on the learning strategies adopted by students [104]. If material is covered as a litany of dates or facts then students will be encouraged (perhaps implicitly) to concentrate on memorising those facts, especially if the assessment focuses on the ability to recall them quickly and accurately. On the other hand, if a subject is taught in a more discursive or analytical way, and this is mirrored in the assessment, students will focus more on understanding.

While learning style is very much a personal attribute of a learner, a teacher can exercise some influence, perhaps by stealth [97], over the learner's activities. This in turn can influence the way in which the learner learns. Students given lists of dates will memorise them, while those engaged in more complex activities will come to achieve an understanding. "To assume that one must teach to a particular learning style misses the point that a given student may be best taught by one method early in learning and by another after the student has gained some confidence" [104].

It is too simplistic to suggest that any single activity will promote a particular type of learning [149]. The same applies to any particular form of teaching [155]. Learning is a highly individual concept and each learner can be expected to take something different from each learning experience. What is important is that all learners are encouraged to engage in the activities that result in the best and most appropriate learning for them.

There is no implied quality judgement here that more complex levels of learning are in some sense better than learning by rote. The way in which learning will be assessed has a significant influence on the sort of approach that is best in any learning situation. If a learner is to be assessed in a way that requires precise recall of facts there is no point in the learner aiming for an understanding.

The following sections describe the most popular models of student learning. It is important to emphasise that none of these models provide convenient pigeon-holes into which a student's preferred learning style can be slotted. Rather they provide a framework for understanding how and why different learners learn in different ways.

2.3.1 Deep and Surface Learning

The classification of learning styles into *deep* and *surface* learning (first described by Marton and Säljö [98]) is well known and is reputed to be the most widely cited work in education. A surface approach is characterised by attempts to memorise material so that it can be repeated verbatim in assessment. Deep learning, on the other hand, concentrates on understanding, which leads to knowledge that can then be applied to new situations when called on or assessed.

It is important to realise that Marton and Säljö did not did not identify generic types of student. There is no such thing as a 'deep student' or a habitual 'surface learner'. They identified learning *styles* and showed how students tend to adopt the style most suited to the assessment at hand (a so-called *strategic* approach). Nor do the styles offer a simple black and white classification – rather they each represent a range or scale of approaches [104].

Ramsden and Entwistle [128] have shown that deep approaches are more likely to be adopted in disciplines that have a fairly low formal workload (notably arts-based courses). Conversely, students are much more likely to adopt surface approaches where the workload and contact time are higher – in broad terms this points to disciplines in science and engineering. They also showed that departmental culture can significantly affect students' approaches. For example a deep approach is more likely to be fostered in an open and friendly teaching department.

It has also been observed that some teachers may tend to teach to a preferred learning style [104] or can tend to teach to accommodate the style that they believe their students prefer. It is a mistake to do this and also a mistake to believe that all members of a class of students have the same style. This approach can lead to a vicious circle of events [156] where students are almost coerced into the learning style expected of them, which in turn reinforces their teacher's expectations.

Some teachers may be tempted to view deep learning as necessarily good and surface learning as always bad. Further they may come to believe that academically able students will always use a deep approach and weak students a surface approach. This is not the case. There are some things that are best learned by rote or simply have to be memorised (for example the alphabet). A student should be encouraged to use the style that is most appropriate to the subject that is being learned. This encouragement does not, of course, come in the form of some florid exhortation to a class to "learn deeply!". It comes from the activities and assessments devised by the teacher. Activities and assessments can be developed that encourage a student to engage in a way that fosters a particular form of learning.

2.3.2 Operational and Comprehension Learning

A similar but less well known view of learning is that developed by Pask [122]. This identifies two learning strategies: the *serialist* (operation learning) and the *holist* (comprehension learning). The serialist approach focuses on detail and often overlooks

the overall picture whereas the holist works on a wider front and looks for general patterns in a problem. The holistic approach therefore makes more use of analogies and previous experience (and is close to Kolb's cycle model). These forms of learning appear, in fact, to be specific forms of the learning styles identified by Marton and Säljö. The serialist is a form of surface learning (with the focus on memorising detail) and the holist is similar to deep learning (with the focus firmly on understanding).

There is no such thing as a student who will always adopt, for example, a holistic approach. While students may tend to prefer a particular approach it is their teacher's responsibility to make sure that they engage in learning activities that promote the type of learning that is most appropriate to a particular learning task.

2.3.3 Engagement

Writing from a firmly constructivist standpoint, Biggs [14] describes the concept of students' *levels of engagement* in their learning and shows how the teaching context can impact on this engagement. The higher levels of engagement are associated with students taking the knowledge they have gained and using it in new situations, while low levels are associated with simple rote learning. The close parallels between these levels and the deep and surface forms of learning are clear. Biggs writes that high levels of engagement promote deep learning and lower levels tend to promote surface learning. The relationship between these levels of engagement and teaching methods is presented graphically in figure 3 on the next page.

Biggs argues that a student's level of engagement will increase as the teaching method used becomes less *passive* and more *active*. Passive teaching consists of activities such as the traditional lecture[6] while active methods include problem-based and discovery learning techniques. He also suggests that this is the case for all students, whether or not they are particularly well motivated to learn the material being taught, or whether

[6] "Traditional Lecture" here refers to that quaint (yet surprisingly popular) activity where one person enters a room and there displays a series of notes for a large number of other people to copy down.

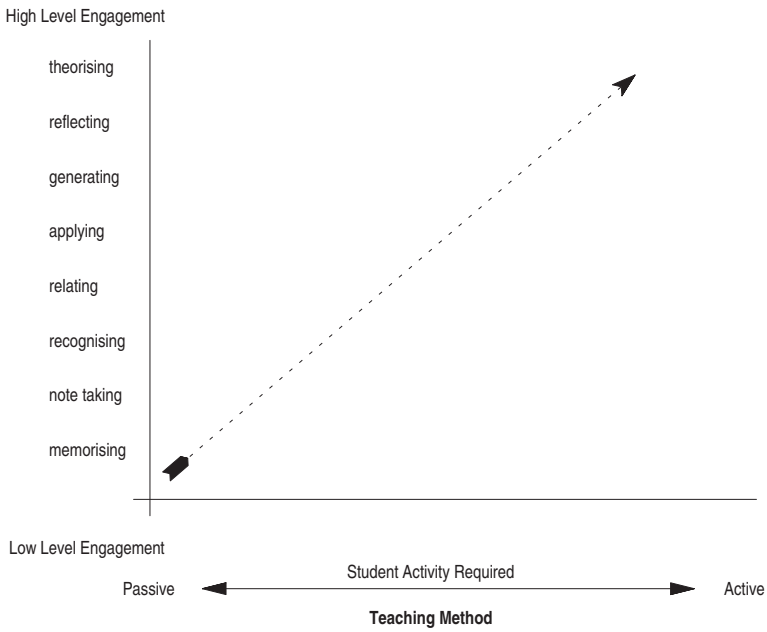


Figure 3: Levels of Engagement and Teaching Methods (adapted from [14])

or not they are particularly academically able. Some students (Biggs calls them "academic") will naturally function at higher levels of engagement (deep learning) even if the teaching is totally passive. They are probably academically able and are interested in the subject. They are well (probably intrinsically) motivated. Their less well motivated (and perhaps less academically able) peers will tend to operate at the lowest level they believe they can get away with and will tend to rely on the simple memorising of notes if at all possible (surface learning). What is needed is active teaching that requires these latter students to engage at a higher level. The two types of student will never engage at the same level but, according to Biggs, the gap in their levels of engagement is the smallest when the teaching is the most active.

Biggs's levels of engagement are similar to the stages in another model – Bloom's taxonomy of educational objectives [15]. Bloom suggests that an individual learner passes through a series of stages, each one of which represents a different form of engagement. This is in contrast to Biggs's model, where a student operates at only one level. Bloom uses a series of six sequential stages to model the way in which a learner develops: *knowledge*, *comprehension*, *application*, *analysis*, *synthesis*, and

finally *evaluation*. This sequence emphasises that learning is a continuous process and not a discrete event.

It follows that a teaching environment and learning activities must be devised that effectively coerce students into a high level of engagement (or to operate at the highest level possible). The teaching methods used and the overall environment and culture (not least that of the teaching department and the institution itself) have a very strong influence on a student's level of engagement. If they are suitably directed all students can learn in the most effective way.

2.4 Conceptions of Learning and Teaching

Students learn in different ways and teachers teach in different ways. Schmeck [137] observes that "the way we go about accomplishing learning will of course depend on what we conceive learning to be". It is easy to see that the students approach to learning (or their level of engagement) will depend on how they view the activity and on their motivation. Similarly the main factor that determines how a particular teacher teaches is that individual's view of what *teaching* is.

Biggs [14] argues that teachers will change in the way in which they think about teaching as they progress through their career. He provides a framework of three levels to describe this development:

1. Learning is a function of differences between students.
2. Learning is a function of teaching.
3. Learning is the result of students' learning-focused activities, which are engaged in by students as a result both of their own perceptions and inputs, and the total teaching context.

These levels are presented in order of complexity and depth, with level 3 the most likely to promote good learning. Biggs's model of career development suggests that

a novice teacher will tend to start at level 1 and will move steadily towards level 3 as a result of more experience, reflection, and professional development.

Prosser and Trigwell [126] present a similar list (written from a less constructivist standpoint than Biggs) of teachers' views of teaching in terms of what they call *conceptions*. These are:

1. Teaching as transmitting concepts of the syllabus.
2. Teaching as transmitting the teacher's knowledge.
3. Teaching as helping students acquire concepts of the syllabus.
4. Teaching as helping students acquire teacher's knowledge.
5. Teaching as helping students develop conceptions.
6. Teaching as helping students change conceptions.

This list is complemented with a further classification of how teachers can view the learning of their students:

1. Learning as accumulating more information to satisfy external demands.
2. Learning as acquiring concepts to satisfy external demands.
3. Learning as acquiring concepts to satisfy internal[7] demands.
4. Learning as conceptual development to satisfy internal demands.
5. Learning as conceptual change to satisfy internal demands.

Both these lists are presented in order of depth. Prosser and Trigwell have also demonstrated that a teacher holding deeper views of teaching is more likely to hold deeper views of learning and vice versa.

[7] This entry and the previous one call to mind extrinsic and intrinsic motivation and seem to imply that the latter is preferable.

The work of Prosser and Trigwell drew on that of Dall'Alba [35]. Dall'Alba undertook one of the earliest studies into teachers' conceptions of teaching and identified seven views (again presented in order of complexity):

1. Teaching as presenting information.
2. Teaching as transmitting information (from teacher to students).
3. Teaching as illustrating the application of theory to practice.
4. Teaching as developing concepts/principles and their relations.
5. Teaching as developing the capacity to be expert.
6. Teaching as exploring ways of understanding from different perspectives.
7. Teaching as bringing about conceptual change.

Dall'Alba also suggests that the deepest conception is "focused on the relationship between teacher, students and content", a systemic view that has clear parallels with Biggs's level 3 and Prosser and Trigwell's deepest levels. Similarly, the shallowest views in each classification are effectively the same, viewing teaching as simply the one-way transmission of information from the teacher to the student for eventual verbatim reproduction.

While they use different terminology, there are clear parallels between these three models. The lower levels in each focus on facts and memorising (surface learning) and the higher levels introduce more complex notions such as understanding. In presenting his model Biggs argues that many teachers will pass through the phases as they develop their teaching. There are many recorded examples of this (for example [116]), although they are not always written in Biggs's terms. It is not unreasonable to suggest that the same is true of the other models and that teachers will also reflect and gradually develop deeper conceptions from the other schemes. It would be unrealistic to claim that there is such a thing as a 'level 1 teacher' who cannot change but there

certainly do seem to exist teachers who see their rôle (perhaps subconsciously) simply as presenting information to students for them to memorise.

The following sections consider Biggs's levels in more detail. This discussion provides a suitable framework for a consideration of how a teacher's conceptions of teaching programming may develop.

2.4.1 Level 1: what the student is

This view is based on the fundamental assumption that some students are in some sense naturally academically able and well motivated while others are not. The basic actions required of students are to attend lectures, to take sensible notes, to learn these for the examination, and to repeat them for the summative assessment. Some students will engage in these activities and some will not. The best students (and Biggs writes that the assumption here is that some students are simply 'best' by their very nature) will engage in these activities, will do additional work outside lectures, and will eventually score the best marks. There is little or nothing the teacher can do to influence the behaviour of the students. In effect the students' success or failure is determined before the course starts. This view clearly corresponds with the lowest views in the models developed by Prosser and Trigwell and Dall'Alba.

This view sees the teaching activity as simply the strictly one-way process of the communication of information (or, worse, the communication of facts). It can be characterised by the old cliché of the transmission of information 'from the lecturer's slides into the students' notes passing through the minds of neither'. Material is 'covered', not taught [156], and the teaching objective is to "transfer the knowledge from the textbook to the minds of the students" [116]. The final summative results are seen as depending solely on a student's application and aptitude.

It follows that student failure is effectively a preordained event. Students have failed because they are bad students, as a result of attributes they possessed before the teaching started. Those students who have done well are simply good students and

were good students before the course began. The teacher's rôle has been merely to communicate the appropriate information (or to cover the required material) – the teacher has had no influence on the students' results. It appears that the teacher is effectively taking the view that both parties are a pawn in the process, both unable to exercise any influence.

Biggs writes that this view of teaching is far too simplistic and yet it remains an easy and attractive explanation (particularly in a course, such as programming, when very many students appear to struggle [102]). But there are flaws to this explanation. It seems reasonable to assume that no-one starts a course at university with the intention of failing and presumably admissions systems (and previous summative assessments) are such that everyone starting the course has the intellectual ability to succeed. It simply cannot be the case that some students are doomed to failure before they start.

This view is likely to be taken by a new teacher presenting a course for the first time. It is a straightforward explanation for the variation in student performance in the class. The students who fail will always fail – the worker blames the raw material.

2.4.2 Level 2: what the teacher does

After some reflection, and probably a few more years' experience, a teacher will move on to this view (and the corresponding middle conceptions in the other models) [31]. The reason for the failure of some of the students is that the teacher (now thinking as an origin rather than as a pawn) has failed in some way or, perhaps less self-critically, that the teacher was teaching in an inappropriate way. This thinking leads to the use of more active teaching techniques. The teacher hopes that these changes will help the students to engage more and so learn better. They may inspire the students or may provide more motivation. The worker seeks some better tools.

The first two levels in Biggs's model attempt to attach the blame for the students' failure to either the students or the teacher. Level 2 thinking shifts this blame from the students to the teacher. Reflective teachers operating at this level will consider

what went wrong in a course. They will collect feedback, analyse it, and from this they will plan how to teach their courses differently next time. After all, it must always be possible to do things better. However, one problem remains. The students are the same and will negotiate the same academic course. It is highly debatable whether or not changes in the teaching methods can make any significant difference to the overall performance of the students.

It is a shame that much of the current research in computing education (especially in programming [75]) appears to be operating at this level. Such research often describes interesting and innovative new ways to present material but there is rarely any compelling evidence of the effectiveness of these approaches.

So the teacher can change the presentation of a course and can introduce exciting new activities, assessments, or assignments. But the teacher soon discovers that there is scant evidence that all this effort has had any significant impact on the learning of the students. After time for some more reflection the teacher will progress to level 3.

2.4.3 Level 3: what the student does

The first two levels in this model are disjoint. They both look to allocate the blame for student failure to one of the two participants in the learning scheme and succeed in nothing more than shifting it around. This final view ignores blame and instead focuses on how teaching activities can promote learning in students. What can be done to make sure that the students learn? More to the point, what can be done to make sure that they cannot avoid learning?

To address these issues a teacher needs to be able to answer various questions. An attempt must be made to define the level of understanding and ability (perhaps skill) that students should reach when they have completed a course. This leads once again to the definition of suitable activities that the students must engage in so as to achieve this degree of understanding and ability (but this time there is a more focused direction to this process). Finally (and this is important – the definition of assessment comes after the definition of the learning objectives) assessment mechanisms can be

devised that will reliably ascertain whether or not this level has indeed been attained.

The three key questions that emerge from this are:

- What skills, knowledge, or understanding should students possess at the end of this course?
- What activities must students engage in to obtain these skills?
- What does the teacher have to do to assess whether or not these skills are indeed possessed?

This view sees the learning process as a combination of many things – as a *system*. The effectiveness of the learning will depend on many factors. These do indeed include the nature of the students but the system also encompasses the teaching activities, the organisational context, and perhaps the course's place in the wider curriculum. There are more – for example the personalities of all concerned and the relationship between the students and the teacher (Where is the control? How is it to be exercised?). This is far from an exhaustive list but it does begin to hint at the complexity of the teaching system.

Biggs's deepest level corresponds with the deeper levels in the other classifications. At this level of thinking the students and teacher are engaged together in what might be called a *systemic relationship*. Teachers are no longer simply communicating the necessary information to the students – the teachers' rôle has become much more that of facilitators. They must facilitate learning by devising suitable activities for the students to engage in and learn from. They must *motivate* the students to engage [75]. The students then become active and reflective learners.

2.5 Theory X and Theory Y

Building on this systemic view of teaching Biggs [14] argues that the climate within which teaching and learning takes place is crucial. Drawing on management science

he applies the theory X and theory Y ideas of McGregor [103] about different forms of trustworthiness in the workplace to an academic environment so as to characterise what ought to constitute a good teaching climate.

2.5.1 Theory X

The basic premise of theory X applied in an academic climate is that the teacher does not trust the students. The underlying assumption made by the teacher is that the students are wholly unmotivated, that they do not want to learn, and that they will cheat if at all possible. They will have to be forced to work. Deadlines will be strictly enforced and regulations relating to such matters as plagiarism will be ruthlessly enforced. This view is characterised by statements such as "I have to give them plenty of assessed work, or they won't do anything at all". This would lead naturally to regular summatively assessed exercises, specified in minute detail and held perhaps as often as weekly. These exercises would be examined zealously for evidence of plagiarism. Taken to extremes the exercises would be reinforced with regular supervised examinations to verify that the students understood the work that they had submitted and that it was thus their own.

A theory X view of the teaching and learning system has clear similarities with the lower-level conceptions presented in the preceding sections of this study. The teacher and the student are not working together in a productive system to the extent that it is almost as if the two are in direct conflict. They certainly do not have the same aim. Control in this relationship clearly lies very much with the teacher.

2.5.2 Theory Y

This is, not surprisingly, the opposite view. Applying the theory once more to an academic setting, a theory Y teacher assumes that the students can be trusted and will work. There is an implicit assumption that students are well motivated. Moreover, it is believed that the students will work better if they are trusted and given more

freedom. In this climate assessed assignments will be fewer (although there may still be the same number of assignments some will just not be (summatively) assessed) and will be loosely specified. There will be more freedom with deadlines. Plagiarism will not be an issue as the students will be trusted not to cheat in assessments.

This view is similar to the higher-level conceptions. Here the teacher is taking much more of a facilitating rôle and is quite happy to let the students learn in their own way. Control has thus passed somewhat to the students. Provided that the students appreciate this, and respond to it, the two parties are working much more together towards a common goal.

There are many reasons why a theory Y approach seems preferable to one based more on theory X. For one thing the climate offered is much more open and friendly and the idea of students happily working away on interesting and loosely specified assignments is very appealing. However, in practice a totally theory Y course would most probably be a disaster. Then again, much the same can be said about a totally theory X course. The ideal climate needs to draw on both but probably draws more from theory Y than theory X.

2.6 Assessment

Assessment has many purposes [20]. It provides feedback, allows a teacher to classify students, provides a formal record for various external bodies, and much more. The classification of assessment into two basic types – summative and formative – is well known. *Summative* assessment provides some (probably numeric) measure of a student's success that is then used in some classification. This typically contributes to some end-of-course score. The main usefulness of this form of assessment lies in the information it conveys to, for example, a future employer. *Formative* assessment is different in that the main purpose is to give students feedback on their progress. This feedback can be far more useful (in the short term at least). Of course, some students will be motivated very much by their scores in the summative assessment

(achievement motivation) and these scores also provide feedback (but often in only a very crude form).

Assessment has a powerful motivational rôle although the relationship is not always a happy one. At the simplest level students who perform badly in summative assessment can quickly lose motivation. Even the way in which results are published (for example whether publicly or privately) can have a powerful effect [135].

Assessment also has a part to play in learning. The feedback from assessment, whether in the form of a simple grade or a more detailed report, can trigger the reflection that is such a vital phase of effective learning. Even a low grade can have merits since it can prompt the students to consider where they have made errors, perhaps to seek advice or help, and hopefully to perform summatively better in the next assessment. At the very lowest level the prospect of assessment can coerce a student into working and hopefully learning. The process of working towards an assessment can be as important, in terms of learning at least, as the grade awarded.

There must be assessment and its positive effects are welcome. At the same time there should not to so much assessment that it detracts from the learning. Above all assessment should not be conducted (or the results reported) in such a way as to risk damaging the motivation of any of the students.

2.7 The Changing Student

The preceding discussion has shown that in an educational setting the teacher and student are working together within a system. They are, however, only a small part of the larger system provided by the learning environment. The physical environment and aspects of the institutional environment or culture in which they operate all have a rôle to play. These three factors – student, teacher, and environment – together form the overall learning system. Things are changing rapidly in the UK higher education system of the early 21st century and the impact of these changes on teaching and learning cannot be overlooked. The following sections briefly discuss some of the more significant changes.

2.7.1 Expansion

In the 1980s about 15% of school leavers went into higher education. Today the number is closer to 40% and there are calls for it to increase beyond 50% [7]. There is no sign of an end to this trend, especially in computing courses. The most obvious result of this is that class sizes are now much larger [110]. At Leeds in the mid-1980s there were some 120 students in the introductory programming class whereas in 2001 there will be over 300 [74]. Expansion brings obvious pressures of scale [32]. More facilities of all kinds are needed, lecture rooms are bigger (and more crowded and uncomfortable), tutors are busier, and support is harder to find. The modern student is often made to feel anonymous [48] and can be just an insignificant name on a list.

Much learning requires easy access to an expert. This is a rôle traditionally filled by the lecturer or tutor. With a large class the teacher is not going to be able to provide time and guidance to all students. Worse, it has been suggested that the culture of anonymity may well mean that students can sometimes feel threatened if such help is offered [48] (although this suggestion is contradicted in a more recent study [59]). Moreover, many staff report [70] that the expansion in numbers has driven them back from innovative teaching (perhaps using problem-based techniques) towards the traditional lecture-based approaches, with assessment designed for ease of marking rather than learning value. Consequently, expansion seems to drive teachers away from deep conceptions of teaching towards lower conceptions and the hoped-for economies of scale.

2.7.2 Diversity

This expansion in numbers clearly leads to increased diversity in the student body [96]. Obviously, universities are now recruiting lower into the 'A' level pool[8] and are thus enrolling less academically able (in terms of previous summative results) students. At the same time, as access widens, there are more students from non-traditional

[8] Either by lowering offers or by ignoring the popular belief that 'A' levels have become easier.

backgrounds, many of whom will have taken a break from studying (and there is government pressure to increase the numbers of students from such backgrounds [8]).

All the students in a more diverse class will be asked to negotiate the same course (it is rarely possible to tailor a learning experience to individuals when operating at this scale [63]). The class will include students who prefer to learn in different ways and who learn better in different ways [77]. This raises the question of whether or not it is sensible to believe that a single course can suit all students. While the answer to this is probably a resounding negative there is seldom any practical alternative for, if nothing else, purely economic reasons.

A more diverse population of students means that students have a greater diversity of motivation for choosing and following the course [114]. Some will undoubtedly still come with a genuine interest in their subject but many (perhaps, in the case of computing, most [28]) may well be more interested in the potentially lucrative career they expect to embark upon after graduation. In 1997 *The Independent* newspaper (quoted in [114]) wrote of what it described as the "disturbing picture" of students who had set aside the traditional student activities of "sex and drugs" and become "ambitious, materialistic individuals". This surely points to a change in motivation.

There is little evidence that these changes have been properly appreciated or reflected in any changes to the ways in which the students are taught or assessed, or indeed in any of the teaching institution's (or wider society's) expectations and procedures. The expectation remains that universities will produce graduates irrespective of changes in the students.

2.7.3 'Full-Time' Students

Today's students are expected to spend the same amount of time on their studies as were their predecessors in the 1980s. Herein lies a problem. The financial conditions under which the two groups operate are very different [9]. In the 1980s few students had part-time jobs, while today most do [10] and many (in computing at least) are

running their own businesses. Students in 2001 need part-time jobs so that they can eat [6] and so the job is often more important to them than their studies (in Maslow's terms it provides the means to satisfy a lower-level need). If students have less time to devote to their course they will do less and it follows that they will learn less. They are also likely to be pushed towards a short-term surface learning approach.

At the same time, and also usually for financial reasons, an increasing number of students choose to attend a university near their home [83]. Some of these students will miss part of the whole culture of university. They will not be members of the community. In practice the students of the 1980s were students all the time whereas today this is definitely not the case.

2.7.4 The Institution

Many aspects of the institution's rules, regulations, expectations, and procedures have failed to react to these changes. It is still assumed that students are devoting all their time to their course when they are not [47]. Academic institutions and departments may bemoan the financial poverty of their students but they seldom change their procedures to alleviate the pressure or show much consideration. Some staff appear to be unaware of the reality of student existence today. They do not make allowances and appear to see the rigorous enforcement of rules and regulations as more important than student learning (theory X).

The present environment in higher education in the UK is very far from conducive to improvements in learning and teaching [85]. With the increasing emphasis on research, and bearing in mind the often uncomfortable relationship between teaching and research [134], teachers often have more work and are under considerably more pressure to produce high quality research. There is thus little time to reflect on their teaching or to plan improvements. The institution itself is driven by many concerns (mostly financial in origin) and the well-being of the students can sometimes be overlooked. While higher education has undoubtedly come a long way from a system

that "mostly worked by the age old method of putting a lot of young people in the vicinity of a lot of books and hoping that something would pass from one to the other" [125], there is still a great deal of need for change.

2.8 Summary

This chapter has discussed various issues and models related to teaching and learning so as to provide a framework for a discussion of the way in which programming is taught (or, equally, the *system* or *climate* in which programming is *learned*). The environment in which students operate in today's mass higher education system in the UK is indeed very different from that of only a few years ago and, perhaps importantly, is very different from that which most of their teachers would have encountered and remember. The effect of this on the students, or on the way in which they approach their studies, is unclear but it is apparent that the cohort is now much more diverse and is faced with many more challenges and pressures outside their studies.

The system within which a student learns in crucial. An environment is required that fosters a high level of engagement and where students engage in well planned learning activities. Ramsden and Entwistle's observation that an open and friendly teaching department is more likely to promote good learning has clear links with Biggs's application of theory X and theory Y to the teaching climate.

The next stage is to take these ideas and to attempt to apply them to the learning and teaching of programming. Programming is an activity with its own distinctive features, so these concepts must be applied with care.

Chapter 3

Learning to Program

First Voice: Come closer now ...

Programming is traditionally taught in higher education by means of a series of lectures. These lectures cover the basic concepts of programming (variables, loops, conditionals, and so on) in some convenient order based on assumed complexity and also on the particular programming paradigm being taught. These concepts are illustrated using the syntax of a particular language and more details of the language are added gradually as the students become more proficient. The students learn from the lectures, from reading textbooks, and from completing various exercises. Some or all of the exercises form the basis of summative assessment.

This model is so widespread that it seems reasonable to assume that at some point in the past it worked or, at least, that the students learned to program (although perhaps not as a direct result of the teaching). Now, however, evidence abounds, in the form of students who simply cannot program, to suggest that this tried-and-tested scheme does not work. What has changed?

One obvious change is the place of computing in the academic world. Computing is still very much a new subject with a short history. In many traditional universities

a computing department has almost evolved rather than having been deliberately created. In the last decades the importance of the subject has increased as it has moved from a niche subject taken by few students to one of the mainstream subjects with almost all students having the opportunity to study computing in some way.

While there have been many changes that have affected all students (outlined in section 2.7) the nature of the students who choose to take computing degrees has also changed. Even as short a time ago as the 1980s those who chose to study this relatively new subject probably had of necessity a strong interest. It is likely that these students had computers at home (when this was a rarity) and had spent many hours programming them. Today, computing is a mainstream and expanding subject with students coming from a variety of backgrounds. There will be some who have spent many years working in the IT industry and others who have never used a computer before. Many of those who have a computer at home will have used it to do no more than run packages and play games – they will rarely have written any programs. The audience to which programming is taught has changed and a first programming course now has to be able to meet the needs of a highly diverse set of students.

Computing is also a degree subject that is assumed by many to lead directly to a lucrative career in the IT industry. This is more than likely to be in the minds of students of today especially considering the poverty that many will endure in order to get their qualification. It follows logically (but this remains to be proven in practice) that many students will embark on the course with the prospect of gaining a highly paid job as the sole aim. These students will have little interest in computing (or programming) other than as a means to an end. Their motivation is very different to that of the computing students of the 1980s.

There is an expanding research literature on the best way to teach programming. Sadly considerably less has been written on the best way to *learn* programming and what there is tends to be in the cognitive psychology literature rather than in that of computing education. Much has been written about the most appropriate language,

environment, and paradigm to use, and the trade-off between choosing a language based on its pedagogical suitability or on the extent of its use in industry. There is an increasing literature on innovative techniques to support introductory programming – suggestions have included the use of visual props [2] such as Frisbees® and Icky Poo®, participative theatre [73], and even singing [144].

These entertaining teaching methods have their place. They serve as what Keller has called an educational novelty and can have a short-term beneficial effect on the students' motivation [81]. They are popular with students, who claim[1] to remember more from the teaching sessions delivered with their aid than from more traditional lectures [111]. Yet there is precious little tangible evidence that they improve the students' learning (although there may be some indirect effect as they may well make students more likely to attend).

The language and paradigm debate is interesting (and at times fierce) but there is little evidence that any particular language or paradigm is best suited for teaching. There exist languages designed specifically for teaching (for example BASIC, LOGO, and Pascal) but few higher education institutions would seriously consider using them due to their lack of industrial application.

Even with these changes and debates the underlying way in which programming is taught remains largely the same. A typical programming course still begins with a consideration of the nature of the task and then introduces programming concepts (in whatever language and paradigm) in sequence. Students are then expected to practise by undertaking exercises. An alternative top-down approach based on data structures and design has been proposed [131] but has not gained wide acceptance.

Lectures are an ineffective way of teaching programming. This was argued as long ago as 1971 (albeit by a psychologist) [154] and it is surprising that it is a method apparently still in widespread use some three decades later. It has also been argued (and this time at least in a computing science education setting) that this approach

[1] They *claim* to. It is unclear how this claim can be measured. This highlights a problem with much research describing innovative ways to teach programming – it is difficult to evaluate.

leads to many difficulties including "passive learning", "premature complexity", and "premature abstraction" [58].

Of course, the best way to learn how to program will be different for different people but any scheme must surely involve elements of:

- *hands-on practice* – programming is a skill and the only way to acquire and develop such a skill is practice.
- *use of reference material* – even experienced programmers will work with a language reference at their side.
- *access to, and advice from, a programming 'guru'* – an effective way to acquire a skill is by an apprenticeship.

The rôle of a teacher of programming is to provide an environment where the students have access to these things and then to ensure that they use them in what is, for them, the most effective way. It is senseless to argue that a single teaching scheme can be equally effective for all learners but this is what most student programmers encounter.

To set the context, this chapter first considers how an experienced programmer would go about learning a new language and contrasts this with the experiences of a novice. The theory from the previous chapter is then applied to the particular case of learning to program and the chapter concludes with a discussion of what precisely it is that makes learning to program so difficult.

3.1 Experienced and Novice Programmers

Before considering and investigating the process (or indeed the processes) by which a novice will learn to program it is useful to start by considering the processes that an experienced programmer would follow in order to acquire a level of proficiency in a new language. An understanding of these processes may well provide the basis of a discussion of the difficulties facing novices as they attempt to learn to program.

Experienced programmers organise their knowledge by remembering a collection of tried-and-tested 'chunks' of program (algorithms, recipes) which they can apply to new programming problems. These chunks (which are sometimes called plans [146] or templates [94]) might even correspond to physical libraries of actual working code ready for re-use. Experienced programmers often, if not usually, create a program as an adaptation of an existing program following a similar algorithm.

Following an essentially constructivist argument, experienced programmers already possess a relevant body of knowledge and all they need to do in order to learn a new language is bring this knowledge to some new situation (in this case the new language). They will go through processes similar to those in Kolb's learning cycle and will adapt their knowledge.

Experienced programmers have relatively simple needs when they come to learn a new language. They simply need to know how their existing mental model of programming translates into the new language. For example at the lowest level they know what a conditional statement is and so just need to know how to implement one in this new language. A reference text will be sufficient, a guru would be useful but not essential, and the rest will come with some hands-on activity. Such learning is often driven by commercial considerations [154] and so may well take place 'on the job'. This process would not take place in the traditional order of a programming course (variables, statements, program flow, procedures, and so on). It is a holistic process of changing, almost translating, existing chunks of knowledge.

Experienced programmers also possess some sort of mental model of the mysterious inner workings of the computer. They will understand at some level at least how computer memory is organised and how instructions are executed. This knowledge, even if it is only at a superficial level, can be invaluable in tracing errors in programs or in understanding why a program does not appear to work. Moreover, much of this knowledge is quite independent of a programming language and it is more knowledge that can be readily mapped to a new environment. In addition the programmers will have some level of understanding of the workings of a compiler – how the program

source is parsed, compiled, and linked to give an executable version. They have learned the rules of the game.

The experienced programmer also has *confidence*. No programmer possessing any significant experience of, say, C++ should be very much daunted by the prospect of learning Java™ for some new project[2] (it has been suggested that the biggest obstacle is likely to be the programmer's own reluctance to change [67]). Once again it is simply a case of taking existing tried-and-tested knowledge and skills and applying them in a new context.

Even if an experienced programmer is changing paradigm (most likely from purely procedural to object-oriented) the change is reasonably straightforward. Most of the mental model and existing skills will translate and what remains to be mastered is a new syntax and perhaps a new design strategy. There are, of course, new skills to be learned (and there may be problems along the way) but any programmer with any significant experience would be expected to master the new paradigm reasonably quickly and with limited suffering.[3]

When compared to experienced programmers novices have many disadvantages when it comes to learning a programming language. It has been shown that novices tend to adopt a very different way of organising their programming knowledge to that adopted by more experienced programmers [105]. Novices have no collection of chunks of code (physical or otherwise) and nowhere obvious to start one. Nor do novices have any sort of model of the programming process [11] to help them, and they often fail to have an overall mental picture of the activity that they are trying to master. Worse,

[2] This is perhaps not the best example. Java is a pure object-oriented language while C++ is essentially a hybrid between an object-oriented and a procedural language (or perhaps a procedural language with some object-oriented features added). Many C++ programmers use the language as what amounts to a procedural language with some additional object-oriented facilities. Such programmers would have to learn to think in a more object-oriented manner when approaching Java and this would represent a not insignificant paradigm shift. Java programmers coming to C++ may well find similar problems in programming in a less object-oriented way.

[3] An interesting side issue here is whether this is a situation where a novice with no prior experience has some sort of advantage over an experienced programmer. A novice learning to use an object-oriented language knows of no other way to think or program – is this a better situation than being proficient in a procedural language?

a novice may have an inaccurate model of the processes underlying programming – a sure recipe for disaster.

Novices generally lack any concrete starting point for their programming. They have never programmed before and so do not have any library of templates or program code from previous experience. The novice is lost without these chunks of knowledge or libraries of programs. A side issue is that the programming process is so natural to the experienced programmer that novices struggling with the same problem can sometimes see the expert in action and quickly become disheartened. They cannot adapt knowledge that they do not possess and it all seems so easy for the expert.[4]

A mental model of the inner workings of the computer is something else that a novice often lacks. Without such a model some of the most basic elements of programming seem arcane and mysterious but with a model they make perfect sense. The process of compilation will seem equally arcane and mysterious as the compiler takes the novice's code and then spits forth error messages that are quite unintelligible to the novice. The novice is playing a game without knowing the rules.

The novice is clearly at a distinct disadvantage in several important departments when compared to the experienced programmer. The challenges facing a novice when learning Java from scratch, for example, are very different indeed from those faced by an experienced programmer already fluent in C++ embarking on the same task.

The process that an expert would use to learn a new language is not going to map directly into a scheme for teaching novices but it throws some light on what novices must be taught or on what they must somehow acquire before they become in some sense experienced. They must acquire the all-important chunks from which they can build programs, they must acquire some sort of mental model of the workings of the computer and of the programming process, and, perhaps above all, they must acquire confidence. Some of these things will be acquired during an introductory programming course while others will come over a number of years and probably only

[4] A good analogy here is juggling. Imagine watching a skilled juggler juggle five balls [141]. The balls are handed over for a novice to try...

as the result of extensive practical experience. Nevertheless it is clear that novices and experienced programmers face very different challenges as they approach a new language.

3.2 The Programming Environment

When devising a system in which a novice learns to program the most important element is arguably the programming environment – the language, operating system, editor, and compiler. There is an immense (and still expanding) literature on the best language for teaching and a smaller, but still extensive, amount on the best platform. More recently this has been extended to include discussion of the best paradigm[5] and whether this ought to be strictly procedural or object-oriented (and in the latter case whether 'objects first', 'objects last', or 'objects at all' [33]). In some cases a functional approach has been proposed but acceptance of this is not widespread.

3.2.1 Language

It is important that a distinction is made (in the minds of both teacher and student) between learning to program and learning a particular programming language. It should always be first and foremost the skill of programming that is being learned. It is to be hoped and expected that a competent student could then easily move to a new language. Nevertheless a novice will naturally have to get to grips with the syntax of a particular language as a first step. There are two basic factors to consider when a language is chosen:

- *pedagogy* – the language's suitability for teaching. Languages that rely heavily on symbolism or neat tricks are unlikely to be suitable for novices.
- *commercial use* – the extent of industry demand for programmers skilled in the language. As many students supposedly see an IT-related degree more and

[5] A hideously misused word.

> more as a first step towards securing a lucrative job they demand to be taught current commercially relevant skills. To some extent the industry itself adds to this pressure.

There are some other technicalities to consider. For example the language must be available on the appropriate platforms, it must support the paradigm to be taught, there must be suitable textbooks, and so on. Nevertheless pedagogy and commercial use are the most important even if they are sometimes at odds.

In 1971 Wirth [159] recognised these issues in his original paper describing the Pascal language. He identified a vicious circle whereby industry dictates the most commonly taught programming language. The most popular language in industry will be the most used and therefore the language most in demand. In response to this demand the most taught language will be the one most used in industry. He deprecated what he called "this stagnation" and as a solution proposed Pascal as a language designed specifically for teaching.

Pascal still has something to offer as a language with which a novice can learn the basic principles of programming. It is a small language but it neatly illustrates all the fundamental concepts of programming using an English-like syntax. A programmer with a sound knowledge of Pascal should be able to abstract the basic principles and move on to other similarly structured languages with a minimum of difficulty.

Current development environments derived from Pascal such as Delphi™ and Turbo Pascal™ offer the potential for interesting and inspiring graphical assignments carried out using an interactive development environment. Of course, the language remains only a small part of the overall system in which a novice learns and the novice will have to be able to abstract the principles learned from Pascal and apply them in other languages, but Pascal would still seem to offer a reasonable setting in which to learn these principles.

However, a survey in 1996 [18] noted that use of Pascal in teaching was decreasing in favour of C or C++. The reasons cited are partly technical and concern Pascal's lack

of support for data abstraction. Significantly, a prime reason cited for the change given is Pascal's "failure as a real world language" but this is a purpose it was never intended to fulfil! Data abstraction is supported in the latest language in the Pascal family, Oberon [51], but the use of this language is also limited by the rush to use industrially popular tools. Anecdotally, Java is today becoming the dominant language used for teaching. This is a change driven to no little extent by the increasing use of Java in industry – Wirth's vicious circle prevails!.

The use of a commercially popular language is undoubtedly attractive to students. Experience at Stanford University [133] was that student enrolment on programming courses increased dramatically when ANSI C (a commercial skill greatly in demand at the time) was introduced as the teaching language. Students certainly seem to show a reluctance to learn anything that they do not see as useful in their future career, so it is perhaps an unfortunate consequence that a commercially popular language will have to be chosen as the basis for teaching programming.

Many institutions teach more than one programming language. There is an initial language, used as a vehicle for the introductory programming course, and then other, perhaps more specialised, languages used in later courses. It was once quite common to learn Pascal initially and then to move on to learn C for more serious programming. This structure (where it is used) means that, in theory at least, the first language taught could be chosen for its pedagogic suitability. Later languages could be chosen for their appropriateness to a real application and would, by implication, be more focused on real world considerations.

This is an appealing idea. Critics would argue that it is foolish to teach one language and then another but these critics would be mixing the teaching of a language with the teaching of programming. Students might also complain about being asked to learn an additional language – it does seem rather like extra work. The increasing use and ubiquitous nature of Java also mitigate against this approach.

It is possible to make arguments for and against all other languages (and indeed all combinations) and it is certainly most unlikely that any particular language or

combination will ever gain universal approval. In essence the problem is insoluble and it seems likely that, for the foreseeable future, educators will be driven largely by the 'flavour of the month' in industry. At present this appears to be C++ or Java although an interesting case has been made for HTML and JavaScript™ [107].

Initially the idea of using HTML as an introductory programming language seems very strange. HTML is just a mark-up language and as such it supports none of the fundamental concepts of programming. However, at first sight the process of producing an HTML implementation of a web page appears to be very similar to that of producing a program (assuming that the HTML is written using only a standard text editor). It is shown in figure 4.

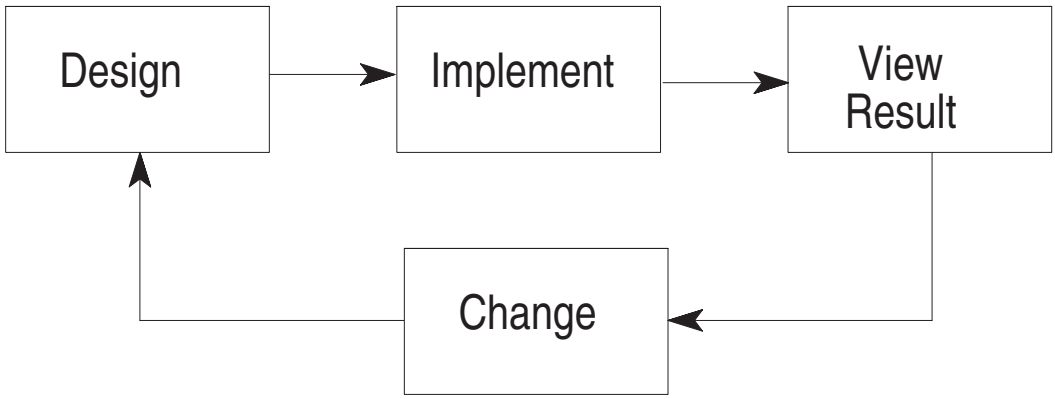


Figure 4: Developing a Web Page

This process – where an HTML file is written, the results are viewed in a browser, and the file is then changed if necessary – is very broadly equivalent to the central part of the programming process where a program file is written, compiled, tested, and then perhaps altered. Moreover, the basic tools that are used (text editors, operating system commands, windowing environments) are the same in each case.

As part of their induction process all the new entrants to the School of Computing's degree programmes at Leeds are asked to complete a short questionnaire about their previous programming experience. Many of the 2000 intake responded to this question with the comment that they had *programmed* in HTML. This raises two issues:

- On entry the students have little idea of what programming is in the context that they are about to learn it.
- They do have some experience, however slight, of an activity closely related to programming that uses many of the same skills and requires much the same overall process.

Perhaps HTML has something to offer as an introductory language or at least as a vehicle for some pre-programming course. The process of producing a web page is similar to the familiar 'edit-compile-test' cycle of developing a program, and this is a cycle that programming students will have to master. While HTML does not support procedural code, JavaScript does. It may be possible (as described in [107]) to use a combination of the two. This issue also starts to raise the question of whether or not programming is most sensibly taught at the very start of a degree course.[6]

However, the idea of using HTML (with or without JavaScript to introduce some procedural elements) to teach programming to mainstream computing students is unlikely to be widely acceptable. It is too drastic a change in practice. For the moment the trend will continue to be to choose from the languages most in demand in industry. It is a huge shame, and an additional unnecessary obstacle for students, that these languages were designed for commercial use by experienced programmers rather than for use by novices in education.

3.2.2 Platform

Novices not only have to master a language. They also have to learn how to use the platform provided for them to develop their programs. This involves learning how to enter their program, compile it, and find the output. Indeed, in their classic book describing C, Kernighan and Ritchie [82] considered these mechanical problems the first "basic hurdle" in learning to program and then optimistically suggested that everything else is "comparatively easy".

[6] This issue will be discussed in more detail later.

There are two extreme approaches in choosing a platform:

- *sackcloth* – using basic operating system-level tools. This will probably be a Unix system using `vi` as the editor and a standard compiler.
- *cosy* – using some sort of interactive development environment (IDE) most usually in a Microsoft Windows® setting.

The choice between these is not obvious. There are undoubtedly many more skills and ideas for novices to master in the first case but the second may well make them feel too secure and may not prepare them for further programming in a less protective environment. In many ways the environment in which programs are written is as important as the programming language itself. For example it is senseless for a programmer to be able to develop using C++ in a particular visual environment if that programmer cannot use C++ on a Unix platform using a standard text editor. At the same time the complexities of using such a Unix environment can often get in the way of the process of learning the language and learning to program when a more visual environment is probably to be preferred.

The ideal choice is clearly somewhere between the two. There are versions of `vi` (for example `VIM` [62]) that can be configured to function as rudimentary IDEs and the use of these can readily support the novice. The choice of Microsoft Windows is attractive not least because it is reasonably safe to assume that most students arriving at university will already be familiar with it. Conversely a Unix platform is more flexible and powerful and provides students with a skill in demand in industry.

Most commercial compilers and program development environments have been, very reasonably, designed for commercial use by experienced programmers[7] (and this also includes most available on a Microsoft® platform). It is surely curious, then, that many of these same tools are often used to teach novices. These systems often produce

[7] Although some vendors have started to expend considerable efforts on marketing some of their commercial development products and environments to the academic community (for example Metrowerks' CodeWarrior® [109]).

error messages that are incomprehensible to the novice, sometimes perhaps because the novice makes errors that an experienced programmer never would. A case in point is the use of a reserved language word as a variable identifier [92] where very few compilers manage to produce a meaningful message (although an experienced programmer would quickly realise what was wrong even without a meaningful error message). Even when an experienced programmer does not immediately understand an error it is normally a reasonably simple matter to find out what is wrong. These systems offer power and flexibility way beyond what a novice learning the language could possibly need and with this power comes more scope for confusion.

A recent innovation has been the emergence of various IDEs developed with the needs of novice programmers firmly in mind. The most prominent of these at present is probably BlueJ, described as an "integrated Java environment specifically designed for introductory teaching" [88]. BlueJ offers an easy-to-use graphical environment that is tailored to the sort of programming and tasks that novice programmers are usually asked to complete. Objects can be created, called, and tested all in the same simple interactive environment, which consists of an integrated editor, compiler, and debugger. This is an attractive option (although only available for Java at present). The criticism might be that BlueJ is not going to be widely used in industry (the same criticism that is levelled at Pascal as a teaching language!) but this is not its purpose. It remains to be seen how widely BlueJ will be adopted in those institutions currently teaching Java and whether similar projects for other languages will emerge.

As new programming languages are developed many are accompanied by claims about their suitability for teaching. A recent example of this is Python with its 'Computer Programming for Everybody' project [151]. Although a relative newcomer, Python certainly seems to have something to offer as an introductory language – it is fully object-oriented and useful programs can be written in a very few lines of code. As with BlueJ it remains to be seen whether or not Python will gain widespread use as an introductory language – it is certainly starting to gain widespread use in the commercial world, which is a promising development.

As with the choice of language the choice of platform is not straightforward. There are advantages and disadvantages to both extreme approaches described here. But both the sackcloth and cosy approaches are extremes and so the ideal choice probably once more lies somewhere between the two.

3.2.3 Language and Platform

The choice of language and platform is not easy. Pascal (or more realistically one of the more recent languages in the Pascal family) remains attractive as an ideal language for teaching novices but commercial pressures mean that its use is unlikely. A similar problem has restricted the use of Ada [18] which promises many of the same advantages as the Pascal family. It has, sadly, to be accepted that the most taught language will be one of those used in industry and that pedagogical considerations are secondary. Wirth was quite correct.

There is more control over the platform. A medium ground must be found between environments that are too basic and too protective. Commercial products do not seem to offer very much since their purpose and market are very different to the needs and characteristics of novice programmers. It may well be that the ideal future platform will emerge from current initiatives such as BlueJ and Python.

Much has been written about the advantages and disadvantages of many languages and platforms. Precious little of this research presents any concrete evidence that any particular environment is better than any other or that there are any significant advantages to be found in any particular combination. Since many institutions teach programming using different environments, and students still appear to fail to learn, the suspicion must be that the programming environment has very little, if any, influence on the effectiveness of the students' learning. The cause must lie elsewhere in the system, perhaps in a less tangible, more psychological, area.

3.3 Motivation

Students will not learn to program if they are not motivated. They must attach some sort of value to learning to program. On one level it matters not why they value learning to program just as long as they do. A teacher might hope for some intrinsic interest in the subject (and many students will indeed have this) but extrinsic or other forms of motivation are equally valid. It is fundamentally important that the teacher appreciates how the class is motivated and manages to tune in to their motivation. This issue will be investigated in chapter 4.

The expectancy-value model of motivation (section 2.1.2) shows that students must also believe that they will succeed in the final assessment of their programming. This is a more personal concept than value and will be investigated in the study of individual students in chapter 5. For the moment it is sufficient to emphasise that motivation is essential if a student is to learn to program.

3.4 Learning Styles

It is very difficult to pin down the most appropriate learning style for the learning of programming. The deep and surface classification (section 2.3.1) seems to work well for a subject which is essentially a body of knowledge but programming is rather more complicated than that. On the one hand it might appear that a deep approach is essential to provide understanding that can be applied in new problem areas. On the other it could equally be argued that programming can be learned as essentially a process that amounts to not much more than simple pattern matching (with the patterns closely corresponding to the expert's chunks of program code) where common problems are spotted and known working sections of code applied. This sounds very much more like a surface approach.

An example illustrates this problem. Understanding, a result of a deep learning approach, should provide the learner with a generic skill than can be applied in

new situations. This might appear to be crucial factor for successfully learning to program where the key skill that is being taught (or learned) – the ability to apply programming concepts and ability to new problem areas – is indeed highly generic. Then again it is perfectly possible to produce working programs without ever fully understanding how they work, perhaps by nothing more than simple trial and error. A programmer might remember that the C++ code:

```
for (int i = 0; i < 10; i ++)
```

produces a loop that executes ten times. It is not necessary to understand how this loop works to use it in a program. While this tactic at first sight resembles a surface learning approach it is also close to the experienced programmer's strategy of remembering chunks of working programs (a decidedly desirable tactic).

Once again it seems that the best strategy for learning to program lies between these two extremes. Surface learning can be useful for remembering syntax or issues such as operator precedence but elements of deep learning (and hence understanding) are crucial if an understanding of the semantics of the language is to be developed together with a genuine competence. This complexity begins to hint at the complexity of the process of learning to program.

The application of Pask's serialist and holist styles (section 2.3.2) is equally confused. Is it essential that a programmer takes an overall (holistic) view of a problem or is the more detailed (serialist) approach to be preferred? It is surely possible to see the advantages of both approaches. Perhaps a more complex approach is needed where the programmer can switch between activities that employ the various styles and take the complementary views offered by each.

If the importance of at least some activities that involve a deep approach is accepted the work of Ramsden and Entwistle has some implications for the environment in which programming is taught. An open and friendly environment with plenty of support is required together with an assessment regime that provides an amount that does not drive the students towards a potentially catastrophic surface approach. In

any case, given the importance of access to a guru, presumably in the form of a lecturer or tutor, an open and friendly environment is essential.

The overall context within which programming is being taught and learned must also be considered. If a surface learning approach is proving to be successful (and is perhaps even being encouraged) in other parts of the students' course it will be difficult for them to work in a different way for their programming. If it is proving successful in *most* other parts of their course it will be very difficult indeed. Students may be tempted to try to memorise syntax without concentrating on an understanding of the underlying semantics. Students do not make a conscious effort to engage in deep or surface learning but habits are hard things to break.

If the learning required is so different it may be that programming is a very different subject to anything that the students have been asked to learn before. This might mean that their tried-and-tested learning strategies break down and are no longer effective.[8] This makes learning so very difficult. The learning strategies that have served a student before (and indeed the learning theory that has identified the learning styles that classify them) do not seem to map to programming.

If the ideal approach lies somewhere between deep and surface learning (or at least combines elements of each) Biggs's diagram of levels of engagement (figure 3 on page 26) provides a promising model with a scale of intermediate points. A reasonable argument would be that the learning of programming requires engagement at least at the applying level and ideally above. A student must learn the required concepts sufficiently well to be able to apply them to new situations and problems. Biggs's diagram suggests that to promote this the teaching cannot be totally passive (and also that it need not be totally active). A teaching environment that coerces the students into a reasonably high level of engagement is therefore required. It also follows that students must be motivated if they are to engage in any learning processes. Recall Biggs's neat definition of a teacher's motivational rôle as "getting students to agree that appropriate task engagement is a good idea" [14].

[8] A similar observation has been made about some problems in geometry [100].

Bloom's taxonomy of learning styles and activities has been convincingly applied to programming [95]. The lower levels correspond to reading, understanding, and interpreting given program code. The middle levels correspond to the development of small programs or code fragments. It is not until the higher levels are reached that complex and original programs can be developed. Since learning to program is seen as a sequential process of passing through these stages there is much to be done before original programs are being written.

Successful students of programming must be able to engage in a range of learning activities designed to exercise a range of learning styles but in a way that is subtly different to the stereotype of the purely strategic student [84]. They must be able to operate at a high level of engagement. The teaching methods used and the overall environment (not least that of the teaching department) have a strong influence in determining the ways in which a student will learn. If teachers can devise a suitable set of teaching methods and can produce a suitable teaching environment they can perhaps hope to motivate the students to study and learn in an appropriate and effective way.

3.5 Conceptions of Learning and Teaching

The preceding consideration of learning styles attempts to describe how a student might be expected to behave when learning to program. This ignores the teachers' side of the system and raises the question of how teachers of programming should view their teaching.

Using Biggs's three levels (section 2.4) it is immediately apparent that the level 1 view is far too simplistic an explanation for the problems in teaching introductory programming. It surely cannot be the case that a significant proportion of the student intake of today can never hope to become competent programmers. It can be argued that the expansion in the number of students learning to program and the resulting diversity of the intake have made the students less well prepared [78]. A related

suspicion (but by no means a unanimous view) is that the lower levels of mathematical attainment and ability of the students makes them less able to learn to program [21] – they have less *aptitude*.

Level 2 thinking in this area brings with it the introduction of innovative instructional techniques. These can serve some purpose but as usual there is little convincing evidence of their value beyond that of simply providing a somewhat diverting novelty.

Programming is a practical discipline. The focus in teaching it should be firmly on what the student does (in Biggs's terms that is level 3). A teacher must devise ways to make sure that the student *does* certain things. A context must be set where the students cannot avoid engaging in certain activities and where they cannot escape without learning something. This presupposes that the students are motivated to do the necessary work but it still seems reasonable to assume that they are (at least at the start). Some current approaches to teaching programming (based at level 2 or even level 1) appear to tend to beat the motivation out of the students with endless assessment. The result of this sort of approach is, predictably, that only the most persistent students succeed.

In passing it is interesting to reflect on how the existing research into introductory programming (and resulting practice in its teaching) at Leeds has progressed through these three levels [75]. While there is limited evidence of any great time spent at level 1 (Leeds recruits students from the higher end of the 'A' level range so they should at least all be reasonably academically able) there has been plenty of effort at level 2 notably with the use of various participative methods [73] and technology-based support mechanisms [80]. In the light of the limited evidence of the success of these approaches it is hoped that efforts are now moving toward level 3 with the focus firmly on the system and on what the student does.

Teachers of programming must, therefore, have a deep understanding and conception (based in whichever taxonomy from section 2.4 is chosen) of their own teaching and of their students' learning. Teachers of programming cannot afford to view their task as the simple transmission of information and they must see themselves much more as a facilitator or motivator [75].

3.6 Theory X and Theory Y

The discussion in section 3.4 started to identify that the ideal academic climate for learning and teaching programming was a friendly and open teaching department. This immediately points to a theory Y model. The theory X and theory Y views both represent extremes – the ideal climate for teaching introductory programming lies somewhere between the two but probably does indeed draw rather more from theory Y than from theory X.

A theory X programming course produces a recipe for stress (for both the students and the teacher). Students will be driven exclusively by assessment. They will learn to complete introductory programming exercises, which is a quite different thing to learning to program. If they retain any motivation it will be to complete the exercises and gain the reward of marks. They are most unlikely to do any additional work on their own. Many will probably cheat, sometimes simply through desperation.

A theory X introductory programming assignment might read:

> *Write a program that accepts a single positive integer greater than one and displays that number in binary. An example executable is available; the output of your program should be exactly the same in every respect. Submit your program before 5pm on Friday 14th April.*

For the teacher a theory X approach would bring with it an immense amount of work in marking programs (and in devising programs that would be easy to assess) and processing marks. This workload is increased further by the task of monitoring for plagiarism and dealing with consequent investigations and punishments. However, a glance at the amount of work published describing convenient automated ways of marking large numbers of identical assignments and detecting plagiarism suggests that this theory X view is not uncommon.

At the other extreme a theory Y introductory programming assignment could be:

> *Write a program that uses all the C++ statements we've covered so far. It can do anything you like. Hand it in some time before the end of semester.*

This appears more attractive at first sight but surely an introductory programming course based solely on the theory Y view would also be a disaster. At the very least it is difficult to see how the teachers (or their superiors or any other stakeholders) could have any confidence whatsoever in the validity of the summative assessment. It would certainly be a less stressful experience for all concerned and teachers would probably find that they have more time to help their students. The element of discovery learning that can be introduced by vaguely defined assignments would motivate and enable students to learn better and at their own pace [3]. Valid assessment remains the stumbling block.

Students will learn better if they learn in an informal, theory Y-style, environment. The potential is there in such an environment to improve their confidence and to promote questioning and discussion [91]. In such an ideal climate the students should want to learn to program. The teacher's rôle will simply be to facilitate this by providing support and a framework. Some assignments will have to be assessed for summative purposes but not all of them need be. All the assignments set should give the students as much flexibility as possible to work on interesting projects. Students are much more likely to want to program if they are interested in or even inspired by what they are doing.

Work at Monash University in Australia [65] has reinforced this by showing how students of programming can benefit from enjoyable informal discussion classes. In these classes the students work in small groups to solve various problems or carry out exercises. The academic content is always present but the informal (theory Y) atmosphere promoted is found to be of great benefit.

The question of the ideal environment for learning to program is based around the notion of trust [75]. The teacher should be able to trust the students to do their best and not to cheat. At the same time the students should be able to trust the teacher to make every effort to catch the students who do still cheat. Students can work on more interesting problems by being allowed to tackle assessments that are loosely specified within some sensible limits. This situation is based more on theory Y than on theory X but is still far from pure theory Y. It is, however, attainable.

3.7 Metaphor

The question of the use of *metaphor* when teaching programming has been raised in passing in a number of the preceding sections. Many of the innovative techniques proposed for teaching programming involve metaphor in one form or another (either in the use of physical props or in other forms of analogy). The discussion of levels of thinking about teaching raises the question of how much the use of metaphor in this way is desirable in teaching programming. This appears to be a level 2 concept, so should it be allowed to intrude on level 3 teaching?

At the most basic level a computer program is a set of instructions that manipulates binary digits. A programmer's task is essentially simply to create a functionally correct set of instructions from a specification. Unfortunately, humans find it hard to think at such a low level and so some higher-level model is needed. This is why experienced programmers use Java rather than machine-level code. At the same time the experienced programmers have some mental model of what the program is actually doing. There is some sort of understanding of registers, program counters, and memory management. They could program at the lower level if they needed to.

Novice programmers on a higher education course that teaches programming as a basic subject have none of these advantages (although somewhat curiously they will probably study the inner workings of computers later on [55]). In particular they lack an understanding of the "properties of the notional machine implied by the language they are learning" [46]. This immediately places a barrier between the student and the teacher to the extent that they have no common terminology, language, or world view. Laurillard's view of teaching as a conversational process [89] is an appealing one for programming where support from a guru is so vital but a meaningful conversation cannot take place between two people who have no common language. The use of metaphor could perhaps help to cross this barrier.

Attempts to achieve this have considered the use of physical props such as Icky Poo [2], acting out algorithms [56], other forms of physical participation [73], and paper-based

models of the computer [100]. All these focus to an extent on the use of metaphor to illustrate some basic concept. Variables can be modelled as boxes and the boxes can have labels (for identifiers) and can be a certain size (a data type). Pointers can be explored using physical means to establish linked lists between students in a lecture room and parameter passing can be illustrated by a deft throw of the Frisbee [2]. Proponents of these techniques report that students enjoy them and later give good course feedback when they are used but there is (as always) little tangible evidence of their effectiveness.

Keller [81] advocates the use of such physical methods as a technique to be used when designing teaching programmes that will motivate. However, he also warns against their over-use and suggests that extensive use will limit their effectiveness and simply confuse the audience. The value of these novelties depends on what Keller calls their "judicious use". There is another danger in the use of metaphor. Novices may come to rely too much on the metaphor at the expense of a true understanding [45]. At some point every metaphor breaks down (even a good metaphor) and the novice must be made aware of when this has happened. Metaphors can be dangerous things, especially if they are relied on.

No less an authority than Edsger Dijkstra [44] has argued strongly against the use of such physical approaches, calling them "infantilization" and accusing those who use them of "contempt of the student body". In a response to the article in which Dijkstra made this comments David Parnas [121] agrees that no teacher of engineering would seriously consider describing "how much the electrons want to get to the opposite side of a capacitor". Dijkstra's suggestion is that programming should be taught as the process of arriving at a formal mathematical representation of a problem using some suitable symbolic language. As it is a mathematical representation it will be possible to test the program for correctness. The final part of Dijkstra's suggestion for teaching programming is that the language used should be one for which no implementations are available. He argues that this would properly emphasise the task of programming as that of producing error-free code.

Even though Dijkstra was setting out his views over 20 years ago his ideas are still relevant today. The higher education system in the UK, and the place of computing in it, has changed almost beyond all recognition, but the fundamentals of computer programming are still the same now as then. Teaching programming using one of today's high-level languages and environments can often cloud or (worse) totally hide the fundamental underlying principles. These principles – the very concepts that would be so closely addressed by Dijkstra's approach – are the key to a truly deep understanding of programming.

Dijkstra's view is extreme and (like much of his other writing on educational matters) probably deliberately intended to be provocative. It does raise a worthwhile caution about the use of metaphor and reinforces Keller's view that it should not be over-used. Metaphor and other out-of-the-ordinary techniques have a place in teaching programming but they are not a panacea. Rather their use adds to the smorgasbord of instructional possibilities for a teacher. Their main purpose should be to aid in the establishing of common ground between the teacher and the learner. After all, the learner will have to come to share the teacher's model at some point and there is nothing worse than having to 'un-learn' something as a metaphor is set aside. A level 3 teacher must not be afraid to dabble with level 2 ideas but must be prepared to stop at dabbling.

3.8 Assessment

Assessment has again been a recurring background theme in many of the preceding sections. To summarise:

- There must be a realistic amount. Too much simply leads to stress and has little summative value anyway.
- All exercises should inspire and should be interesting.
- All exercises should incorporate some flexibility to allow students to work on

tasks that they enjoy.

- A more relaxed theory Y attitude to deadlines and procedures is desirable (but should not be taken to extremes).

A programming course has two distinct objectives. From the teacher's perspective the most important intention of the course is hopefully that the students will learn to program. This aim can be characterised as a *learning* objective. At the same time the students must be assessed and allocated a grade for that part of their degree. This is an *assessment* objective. Many students will quite naturally regard this second objective as the more important of the two.

It is not uncommon for these two objectives to conflict [49]. For example a student will be expected to complete various summative assignments during a course. If one is found to be particularly difficult and results in a low grade the student may well seek help from the teacher. The teacher then has a dilemma. It is possible to teach this student to program but it is also possible to provide coaching to a stage where the assessment can be successfully passed (in a summative sense at least). The teacher should value the first of these options more highly. The problem is that the student is unlikely to engage (or be motivated) unless convinced that the assessment will be successfully negotiated [49]. The student will thus be more interested in the teacher's second option.

As long ago as 1938 Dewey wrote that "all genuine education comes about through experience" [43], a statement that neatly sums up the constructivist view. It follows from this that the only way to learn how to program effectively is for the learner to gain as much experience as possible of writing programs. This process must not be driven by assessment (or at least not by summative assessment). A theory X model where students are given weekly tasks to complete for summative credit ("or otherwise they won't do it") is a sure recipe for disaster. Theory Y would also bring chaos but of a different type. A middle ground is needed. As students learn at different speeds (and will have different pre-existing skill levels) surely it makes sense to leave all summative assessment until the end of the course.

Assessment can also be a powerful motivational tool [135]. It is perhaps idealistic to suggest that students would still engage in programming exercises even if they were not being assessed in some way (even if only formatively) but they should be given the chance. When and if this approach fails more summative assessment could be judiciously and sparingly applied to make sure that the students are suitably motivated.

Introductory programming courses often fall back on mathematical examples (largely because this is all that seems possible with the basic programming concepts and constructs that have been covered in the early stages of the course). This material is hardly going to inspire many students. If mathematical examples have to be used it is surely more interesting to use real data from real life applications such as real NASA data from the Viking Orbiter [54] or something else similarly interesting. Even basic algorithms can be explored within the context of encryption [68]. Perhaps the entire programming process can be set in the environment of a game [112] or anything else that might maintain interest.

To take this idea further students can be allowed to define their own assignments, provided that the end results meet some set of criteria specified by the teacher [61]. The teacher provides some sort of framework, probably by giving a list of features that must be included, and perhaps vets the students' ideas to make sure that they are reasonable and possible with the language being taught. The students can then write a program in an area that interests them to show off the programming they have learned (as an aside, this also neatly defeats the problem of plagiarism). Surely this will be more interesting.

The counter-argument is that the reliable assessment of such exercises is difficult. Indeed it is, so the exercises should be assessed only formatively. The final summative assessment can be specified as tightly as required, as is usual in formal examinations. Summative assessment is necessary but zeal in its application must not be allowed to get in the way of the real business at hand – the learning.

3.9 The Difficulty of Learning to Program

Programming seems to occupy a particular place of prominence in the first year of degree-level computing courses. It seems to dominate the students' experience to the extent that those teaching other courses often complain that the students are spending all their time programming. No-one with any substantial experience of teaching programming would claim that the students find it easy. Some students seem almost to want to hide from programming as they retreat, mollusc-like, into their shells in an attempt to deny its very existence.

It is sometimes argued that the students who find programming difficult are simply those for whom programming is difficult. The argument is simply that some students have no aptitude for programming. The required skills often cited are problem-solving ability and mathematical ability but there is little evidence that either has any significant effect (although a recent study in Ireland [21] has once again hinted at some connection between programming aptitude and experience in mathematics and science). An attempt to address the diversity of the class at Leeds [77] employed simple aptitude testing aimed at these two skills but the final results of the course showed no significant correlation between the calculated aptitude and the final grade.[9]

It certainly helps to have some previous experience in programming before starting a programming course [64] but this is not a measure of aptitude for programming as such. There exist programming aptitude tests (PAT) produced by IBM but the evidence for their effectiveness is at best inconclusive [101]. Other work [52] appears to have shown in addition that no demographic factor is a particularly strong indicator of likely success in programming.

The focus must then turn to cognitive factors such as learning styles or motivation. It has been demonstrated that students who fail programming courses are likely to be more extrinsically motivated than their peers who excel [143] but there is no study

[9] More recent work on the diversity of the programming class at the University of Southampton [36], building on that at Leeds, appears to show that the students' own assessments of their feelings about the course is a reasonable predictor of their final performance.

that shows anything more than possible links between success in programming and preferred approach to learning or observed learning style.

Perhaps the root of the problem lies in the subject itself. Is it the case that there is something inherent in programming that makes it difficult to learn? The following sections consider some of the reasons why this might just be the case.

3.9.1 Multiple Skill

Programming is a complicated business. An experienced programmer draws on many skills some of which have little obvious relevance to the process of producing working code for a computer. These skills are depicted in figure 5.

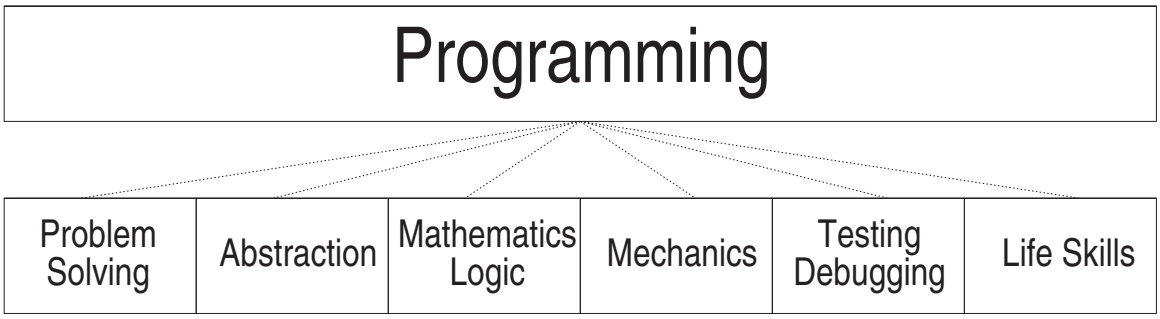


Figure 5: Skills Required for Programming

Problem-solving ability is essential. A programmer must be able to devise ways of making the computer solve a particular problem sometimes in a particularly neat or efficient way. This involves *abstracting* a generalised representation of a problem from a specification. This expression must be made in logical terms (*if ... then ...*, *while ... do*) using elements of discrete *mathematics* and *logic*. A knowledge of the *mechanics* of producing a program (editing the file, compiling, finding the output) is essential. The ability to *test* a program thoroughly to find and correct bugs covers the final stage of the programming process.

Figure 5 also includes *life skills* which is appropriate only for students. Experienced programmers do indeed require some life skills, of course, but the skills are rather

different. Programming is normally taught as a fundamental subject at the start of an overall programme [55]. This is a difficult time for many students. It is a time of transition as they adapt to life and study at university and this adjustment requires that they learn many things. At this point their expected enthusiasm to succeed may well come into conflict with a fear of losing control [132] resulting in very high levels of anxiety and distress. A programming course will bring many more new challenges, both academic and personal. Students will have to learn how and when to seek help, they will have to learn how to manage their own study and social time effectively, and they will have to find their place and make new friends. These are difficult enough things to do when a student is well settled but even harder at a difficult and stressful transitional time.

The skills required for programming are not applied in isolation. They are applied in the context of a particular problem or problem area. A programmer must understand the problem and probably some aspects of the domain in which the problem exists before these skills can be applied. This introduces even more skills into the picture. The precise nature of these skills will be different in each case and so the programmer has to be flexible.

Programming, then, is not a simple single skill (and it is certainly not a discrete body of knowledge). To become a competent programmer a student must master many things. Some of these are taught at the same time as programming, some may not be explicitly taught (for example a teacher may assume that students will simply pick up the use of an editor), and some are probably regarded as outside the direct remit of a teaching department. To compound the problem (and somewhat curiously) some of the skills will be taught *after* programming. Nevertheless, a student must master them all.

Of course, some of these skills are required by all students no matter what subject they have chosen to study. All students need to develop their life skills during their first few weeks at university so that they are able to adjust to life in a new environment. Many subjects require problem-solving ability and skills in mathematics. The important

point is that programming students require all these skills and so find themselves in an unusual situation.[10]

3.9.2 Multi-layered Skill

Figure 5 (on page 69) showed the skills needed for programming in a flat structure. In reality these skills form a hierarchy (or chain [145]) and a programmer will be using many of them at any single point in time [112]. When faced with learning a hierarchy of skills a student will generally learn the lower-level skills first before progressing upwards [12]. In the case of coding this implies that students will first learn the details of syntax, then semantics, then structure, and finally style. Teachers will be familiar with students who produce programs with no indentation, intending to add indentation once the program works, or with no modules, intending to split the program when it works. No experienced programmer would do these things but these are bad habits that it is hard to leave behind (as well as habits that make programming and debugging even more difficult).

It has been shown [94] that novice programmers tend to concentrate on syntax and that this approach is reinforced by the way in which the topic is presented to them in both lectures and in textbooks. This approach leads to an unwieldy collection of knowledge and often almost random attempts to form the remembered syntax into a working program. Without a knowledge of the higher-level issues of algorithms students can never hope to write a program in any sensible and structured way. Programming requires the mastery of many skills and it is far from obvious which is best learned first.

From the teachers' perspective it is surely very difficult to devise a lecture that can deal with skills or information at several levels. For example lectures that focus on details of syntax cannot really address the higher-level skills of understanding [19]. This multi-layered aspect to the programming skill is not merely a problem for the learners.

[10] A first year student at Leeds once pointed out that the only other discipline that required all these skills in such profusion was Medicine.

3.9.3 Multiple Processes

The process of transforming a specification into working program code is not linear but probably consists of three distinct transitional phases between four identifiable stages (figure 6).[11]

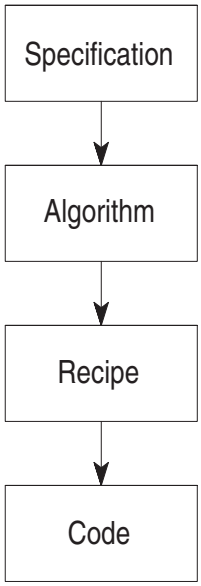


Figure 6: The Process of Programming

Starting with the specification (normally written in plain language) the programmer must first draw on an understanding of the problem domain to devise an appropriate algorithm. Loosely, this involves the programmer in re-writing the specification in a precise way that is closer to an implementation. This is the hardest part of the task and is something that draws heavily on abstraction skills. The final algorithm is then translated into programming concepts or building blocks. This is a simpler task for an experienced programmer who has probably seen similar programs before and has a mental library of code chunks. Finally this design (or recipe) is translated into actual program code. Given a correct recipe this final stage is trivial and is the only one completely determined by the programming language being used.

[11] Obviously an experienced programmer would not pass through these phases in a particularly linear manner. This admittedly simplistic view is just intended to illustrate the fact that programming requires the application of several processes.

A student must master these three distinct phases. It is therefore unfortunate that teaching can often concentrate on the low-level minutiae of syntax (something that is perhaps encouraged by the use of highly symbolic languages and that is definitely encouraged by many textbooks) at the expense of the higher-level, more complex, processes of devising an algorithm. The issue arises of whether or not it is sensible to teach syntax before the whole process has been understood and practised. This recalls Dijkstra's suggestion that programming should be taught in an environment where the programming language is not implemented. There certainly seems to be very little point in lecturing on syntax to students who do not understand how to generate an algorithm from a specification.

Students will also tend to learn syntax first. They might therefore be expected to have some chance with the third transition (given a correct design) but they will never get there without some skill in the first two transitions. Again this calls to mind the student hopelessly trying to compose a program by combining what appears to be completely random syntax.

3.9.4 Misleading

A novice may well at first find the entire concept of programming alien. However, hidden within this strange world will be some things that look familiar. An example might be the everyday concept of 'or' [147] which has a linguistic meaning as well as its meaning in Boolean logic and therefore in a programming language. Drawing on this apparent analogy a novice's attempt to code the statement 'if the answer is y or Y' in Pascal might well be:

```
if answer = 'y' or 'Y' then
```

This statement can make perfect sense to a novice programmer because when read aloud (*"if the answer equals a lower case y or an upper case y"*) it appears to be exactly what is required. Worse still is the C++ version of this:

```
if (answer == 'y' || 'Y')
```

While this is almost certainly semantic nonsense it is syntactically valid, will compile, and will appear to work in certain cases.

An experienced programmer would immediately suspect that both these examples are nonsense and would never consider writing such code. This highlights another problem in the teaching and learning relationship. Teachers have to learn to trace errors that they themselves (as experts [13]) would never make. Dijkstra [44] suggests that this lack of a concrete everyday *analogy* to the programming process is one factor which makes learning to program so difficult.

This also raises the problem that programming as an activity is far removed from much day-to-day activity and hence prior experience. The basic building blocks and tools of programming have no counterpart in day-to-day experience in any meaningful sense and, worse, the nearest parallel worlds – arguably mathematics and language – contain many instances of this negative transfer [112]. Students writing code such as the Pascal and C++ in this section have simply transferred the semantics of a linguistic 'or' to the domain of programming. It is not unreasonable that they find this misleading.

3.9.5 Educational Novelty

Dijkstra [44] takes the view that learning is in general the process of transforming the "novel to the familiar" in a slow and gradual process based on metaphor and analogy (constructivism again). He argues that programming represents a "radical novelty" in which this comfortable tried-and-tested learning system no longer works. Further, adjusting to such a radical novelty is an unpopular activity because it requires a great deal of hard work. The crux of the problem, according to Dijkstra, is that radical novelties are so disturbing that "they tend to be suppressed or ignored to the extent that even the possibility of their existence ... is more often denied than admitted".

A feature of programming is that it is "problem-solving intensive" [123] in that it requires a significant amount of effort (in several different skill areas) for often a very small return. At the same time it is "precision intensive" [123] meaning that the modest success that can be achieved by a novice requires a very high level of precision and certainly a much higher level than most (if not all) other subjects.

Dijkstra agrees with this and observes that a feature of programming is that the "smallest perturbation" [44] of a single bit in a program can render the program totally useless. This has a lot in common with mathematical proof (and this is the direction from which Dijkstra comes) and many engineering disciplines but it highlights the difficulty. There are few everyday situations where this level of precision is required.

Students who start a programming course in higher education have come from their own familiar academic setting. In this comfortable environment they were studying topics with which they were, on the whole, happy and familiar and which they had been studying for some years. They were most likely used to performing quite well academically and had developed a set of tried-and-tested study skills and approaches to learning. To arrive in a setting where they are confronted with a totally new topic (and one that does not respond to their habitual study approaches) must surely represent a radical novelty in Dijkstra's terms. It is perhaps no longer study skills that are required but coping skills.

3.9.6 Language

Any teacher of programming must be sure to draw a clear distinction between learning to program and learning a particular programming language. Most teachers would agree that the purpose of an introductory programming course is the former with the latter providing no more than a convenient vehicle. It is hard for students to appreciate the higher-level, more abstract, concepts of programming while they are being taught to deal with the idiosyncrasies of a particular programming language. This level of understanding comes only when programmers becomes proficient in many

languages and can reflect on their learning so as to pick out an overall pattern and the more general context. It is only then that someone can truly claim to have learned to program.

3.9.7 Interest

Learning (or perhaps here 'being taught' is better) programming can be very dull. Lectures covering the details of syntax are never going to be especially inspiring and exercises that involve simple mathematical manipulations of collections of student marks, stock levels, baseball statistics, or bank account details can never claim to be interesting. At its best programming can be an enjoyable and creative occupation but this only comes when a certain level of proficiency has been attained. It is not difficult to see how a novice programmer could quickly form the view that programming was simply rather boring.

Several teachers have published work describing ways to introduce more interesting exercises into programming classes (for example [54], [61], and [112]) and also work on approaches to make the presentation of programming concepts more entertaining and dynamic (for example [2], [73], and [144]). It is claimed that these ideas make the learning of programming less dull but, sadly, there is as usual little concrete evidence to suggest that the students learn any better when exposed to such innovations. Still, any attempts to make learning to program more interesting must be encouraged. Students will not engage if they are not inspired.

3.9.8 Reputation and Image

Programming courses acquire a reputation of being difficult. This view is passed to the new students by their predecessors and is often exaggerated in the telling. This perhaps makes it acceptable, even expected, that a student will have difficulties in the course (hinting at learned helplessness). At the same time there is the popular image of a programmer to consider. This is of a socially-inadequate 'nerd' who spends all

hours of the day and night producing arcane and unintelligible code (they have been called "code warriors" [153]). It is hard to imagine many students aspiring to this image.

If students approach a course with an expectation that it will be very difficult, and with a negative image of those who excel in the subject, it is very hard to imagine them being especially motivated.

3.9.9 Peers

Students approach a programming course from a wide variety of backgrounds [77]. Some will not have used a computer before (although admittedly this number is shrinking dramatically) while others will have worked for many years as professional programmers. The presence of this last group can in some cases have adverse effects on the progress of the novices.[12] It is hard enough trying to learn to program and the presence of others who find the process so easy and natural can have a seriously negative effect on motivation.[13] Novice programmers may begin to feel inadequate or feel that programming is simply something they cannot do (learned helplessness again). This feeling is often sadly reinforced by the questions that some experienced students will ask in lectures (often simply, it has to be suspected, as a means to expose their own abilities to the whole class).

The students' teacher will also make programming appear easy in a lecture setting and during practical demonstrations. Novice students often report that they find lectures easy to follow and can understand the programs that they see there. However, when it comes to applying the higher-level skills [12] and writing their own programs from scratch they are at a total loss. They cannot understand how something that

[12] Hence the work carried out at Leeds [78] and Southampton [36] intended to remove these students from the mainstream teaching into alternative classes more suited to their experience.

[13] Working in groups can, of course, be beneficial. Hence the work at Leeds and Southampton again which also aimed to group students of equivalent experience and background together so that they can find support and encouragement from others with similar expertise working at a similar level.

appears so easy to someone else is totally impenetrable to them. Their motivation and confidence may well be increased by the experience of following a programming lecture but the later inability to translate this understanding into practical programming is a sure recipe for lowered motivation.

3.9.10 Pace

In an academic setting programming is taught (and therefore learned) to some sort of predetermined set timescale. It matters not whether this is one or two semesters, or even a number of years, the assumption remains that at some point the programming course will end and the students who pass will, presumably, be able to program. This means that the pace of instruction is not under the students' control [93] and it is more than likely that in the class there will be students who learn at different paces. This will lead to the situation where novices cannot follow an example illustrating some new concept because they do not understand some already-covered material. It is easy to see how this could lead to feelings of helplessness and disillusionment or specifically learned helplessness.

3.10 Summary

Programming is certainly a complicated skill to master and learning to program is correspondingly complex. There are many features of the skill that cause this but it must be possible to overcome them (or there would be no experienced programmers!). Since it is a skill, parts of the generally accepted theory of learning do not transfer especially well. This highlights the essential problem that programming is indeed an *educational novelty* for all who approach it. Learning it will therefore be difficult and probably unpopular. If learning to program is going to be a struggle the motivation of those who learn it will be even more important than it is with a more conventional subject.

This chapter has considered some aspects of the environment in which students learn to program. They need to acquire the skills possessed by experienced programmers and a teacher's primary rôle should be to facilitate this. It is probably safe to assume that the students are reasonably well motivated before they start to learn to program (but this will be investigated in the next chapter) and arguably it does not matter particularly what their motivation is. Nevertheless, this motivation is crucial and must be maintained. Its nature and development are the focus of the next two chapters and of the remainder of this work.

Chapter 4

Value – The Class

Mog Edwards: . . . where the change hums on wires . . .

The first stage of the empirical part of this study examines the overall views and motivations of a cohort of students as they follow an introductory programming course in a traditional UK University. The aim is to gain an overall impression of the motivation of a class of students studying an introductory programming course and of the development of that motivation as the course progresses. In the terms of the expectancy-value model of motivation the focus here is primarily on the value component. Why do these students attach value to success in this course?

The classes surveyed were those at the School of Computing at the University of Leeds and at the Computing Laboratory at the University of Kent at Canterbury. The two institutions recruit students from roughly the same pool (the main difference is geographic with Kent taking more students from the south of the country and vice versa) and have comparable 'A' level entry requirements. The main difference in the teaching of programming is that Java is taught at Kent whereas Leeds teaches C++. Nevertheless, for the purposes of this investigation the two classes were treated as a single cohort. The students included were all those taking the programming course

in the first year of their degree programme. These included both those following single-subject and joint-honours programmes. All the students were studying for first degrees. The differences in the backgrounds and learning experiences of the students will be considered in section 4.1.3.

The students were surveyed by brief questionnaires presented in lectures at three key points in their course. The first questionnaire was presented at the start of the course before any programming had been taught and the others followed at the midpoint and finally at the end. For the purposes of this study and in the interests of simplicity the whole first year programming stream is treated as one course (in reality it is two separate modules at Leeds and a part of one module at Kent). The questionnaires used are included in appendix A.

A somewhat similar study to that described in this chapter was carried out in 1998 at Middlesex University [34]. This work surveyed students taking programming across the first two years of their course (and so is not directly comparable to the present study) but its results are nevertheless interesting. The survey was also carried out by questionnaires which first asked the students to respond to a free-form question to express their motivation and secondly asked them to select from a predetermined list to express the same.

At the start of the course at Middlesex four motivations clearly dominated in the answers to the unprompted part of the survey. These were:

- to learn to program.
- to become a professional programmer.
- an understanding of programming will help in a future career.
- to understand more about programs and programming.

Over the next two semesters these remained dominant with the proportion selecting all but the second appearing constant. The exception, 'to become a professional programmer', decreased in popularity from some 20% to a negligible 2% during the

lifetime of the survey. At the same time a fifth motivation 'because you have been told to do it' increased in popularity from only 1% to 19% (another 'to get the credits' also increased from $1\frac{1}{2}$% to 9% over the same time). This clearly points to a significant change in the students' motivation from *extrinsic* and *intrinsic* factors to what might well be classed as *null* motivation.

When the students were asked to select from a predetermined list two motivations dominated throughout:

- to learn how to program.
- an understanding of programming with help in your future career.

The first of these choices points to intrinsic motivation while the second indicates more extrinsic factors. Intriguingly the number selecting 'because you have been told to do it' from the predetermined list remained small throughout in this survey.

The final conclusions from the work at Middlesex stated not unreasonably that the majority of students approaching a programming course see some future career in programming as an attractive possibility and are, therefore, (in the terminology of the present study) extrinsically motivated. This interest decreases as they progress through their course. The students' motivation changes and gradually tends to null factors. It will be interesting to see whether these findings are reproduced at Leeds and Kent.

4.1 Methodology

The surveys were carried out by means of a series of three short questionnaires (shown in sequence in appendix A). There were potentially over 400 students to be surveyed and this method was expected to produce the greatest possible rate of return. With many potential responses it was not really feasible to rely on other techniques such as interviews or focus groups. The questionnaires were distributed in lectures in order to maximise the coverage.

One question was included on all the questionnaires in order to identify the dominant motivation of the class (as defined in section 2.1.1). It asked the students to choose from a set of statements the statement that which most accurately described their motivation for their degree course:

1. I want to do well for my own satisfaction.
2. I want to do well to please my parents or family.
3. I want to do well to please my teacher.
4. I want to do well so that I will be able to get a good job.
5. I just want to pass.

These answers guide students to identifying their main motivation (in categories that correspond to those already described in section 2.1.1) but without making explicit mention of them. The students chose the most appropriate statement simply by ticking the box next to it.

These five statements correspond to the types of motivation already identified and categorised as follows:

1. *achievement* – the sense of doing well.
2. *social* – pleasing those whose opinion is valued.
3. *social* – this time to please another figure[1].
4. *extrinsic* – rewards from a future career.
5. *null* – no particular motivation.

[1] Two statements were used for social motivation because the possibility of a student wanting to please their teacher seems sufficiently different to a desire to please their family (and much less likely).

It was expected that evidence of the final category of motivation (*intrinsic*) would emerge in the answers to other questions on the questionnaires although achievement motivation is effectively a particular form of intrinsic motivation[2] (deriving from the results of learning in its own right rather than directly from the subject itself).

The questions on the remainder of the questionnaires were different for each and were chosen depending on the point in the course reached (they are described in the appropriate sections). Where open subjective responses were required, they were always asked for in the form of a single word. It was hoped that this would be preferable to a selection from a list which might guide students to particular answers or to the answers they might think the investigator wants to hear [118] (a possible cause of some of the rather incongruous findings from the Middlesex study). The words can be sorted, categorised, and sets of words taken to indicate the same essential motivation (for example 'money', 'career', and 'salary' would all indicate that the motivation lies in the expected financial rewards from some future career). The choice of categories can also be guided by the words given to allow a highly unrestricted analysis. The following sections describe the categories that emerged and provide examples of the words assigned to each. A full list of the words received for each of the questionnaires is included as appendix B.

The questionnaires were anonymous and included only basic demographic details from each student. These details allowed analysis by different institutions and degree programmes. It would also have been possible to analyse the results based on gender, age, or country of origin but such analyses are beyond the scope of the present study.

The results should be interpreted in the light of the differences in the way in which programming is taught at Leeds and Kent. It is also recognised that distributing the questionnaires in lectures will restrict the study to responses from those students who actually attend lectures. In particular it is to be expected that the later surveys will have a lower return rate as more students are absent from lectures. Nevertheless,

[2] A statement directly addressing an interest in computing was not appropriate due to the range of joint-honours degree programmes included.

the 'in-lecture' approach offers the highest possible return rate especially as it was arranged that the responses would be collected in the same lectures as they were distributed.

Before presenting the results of the questionnaires the following sections describe the programming courses and Leeds and Kent and consider whether or not there are any significant differences that may affect the results.

4.1.1 Programming at Leeds

The School of Computing at the University of Leeds is one of the largest university computing departments in the UK. In the 2000/01 session there were approximately 800 undergraduate student FTEs[3] and 32 teaching staff FTEs. The School offers four single-subject degree programmes and has a significant stake in many joint-honours degrees. The past few years have seen a significant increase in student numbers at a rate well above the overall figures for the University.

The four single-subject degrees cover rather different aspects of computing and so provide a broad range of options. *Computer Science* takes a very theoretical view, while *Information Systems* is based much more on applications and human issues. *Computing* has recently been introduced to provide a third option in the middle ground between these two. Finally, *Cognitive Science* gives students the opportunity to study artificial intelligence together with courses in psychology and philosophy (as such it is very much more like a joint-honours programme than a single subject).

All degrees run for three years although many students choose to take an optional placement year between the second and final years to make a four year course. The degree programmes are modular with each year consisting of 120 'credits', normally made up of twelve equally weighted modules. Most modules are taught over twelve weeks either before or after Christmas. There are usually two lectures for each module

[3] An FTE is a 'Full Time Equivalent'. One single-subject student is one FTE and a joint-honours student is less (the proportion depending on the amount of their time that they spend studying computing). A similar definition applies to staff FTEs.

each week and these may be backed with examples classes. University-wide rules require a student to spend 75 hours on each module. In addition to their module commitments students are expected to attend a one hour small group tutorial with their designated Personal Tutor every week during their first year.

Programming is fundamental to all the degrees in which the School is involved. At the introductory level it is taught in two modules over the first year. In the year of this study the first semester module covered a largely procedural subset of C++ and made some use of a small number of STL features for simplicity. This was followed in the second semester by a module covering object-oriented C++. These two modules, together making up a sixth of each student's time, would be the only programming studied by most in their first year (a few would study some basic Prolog elsewhere). Some students will learn more C++, and possibly Java, in the second year of their course. A few will study some functional programming in ML. All programming modules use a Unix (specifically Linux®) platform with a `vi`-compatible editor. The C++ compiler used is the Free Software Foundation-supported `g++` from the GNU Compiler Collection together with the `ddd` debugger (actually a graphical interface to GNU's `gdb`).

The class size for introductory programming is around 300. The module is taught by two members of staff and the majority of the formal teaching is by the two lectures each week. In addition a supervised lab session [38] is provided where students carry out a practical assignment with immediate support available.

Both modules are assessed completely by coursework (there is no examination). There are four assessed exercises backed up by two 'Validation Tests' on the same material. These tests are largely intended as devices to detect plagiarism and are thus designed to encourage the students to engage with the coursework material rather than relying on the efforts of others.[4] All coursework is carried out by the students working alone

[4] The consequences for an individual of high coursework marks and low marks in the tests are unpleasant since this phenomenon is generally taken to cast doubt on the legitimacy of the process whereby the coursework marks were achieved.

but each exercise is set at a series of levels so that each student may work to a chosen level depending on ability or inclination [38].

A matter of some concern in recent years has been the apparent diversity of the class [77]. A first attempt to address this has been the establishment of a 'fast track' group of students. These are taught separately on the basis that they have significant prior programming experience. They have only one formal timetabled hour each week and follow a different assessment regime. The benefit here is that a number of experienced programmers are removed from the main class and this hopefully makes that class more homogeneous. At the same time these fast-track students benefit from working with others of similar experience.

The School of Computing is proud of its friendly atmosphere and students are always encouraged to seek help directly from staff. Students are also frequently urged to seek advice and additional help and support if they find the programming module particularly difficult. The module staff generally form 'extra help' groups of these students who are given the opportunity to attend an informal extra hour's tutorial each week. Experience has shown [22] that, for a variety of reasons, these additional classes are taken mostly by the female students so other support mechanisms are in place to address the needs of the male students who appear to tend to prefer electronic and impersonal support. Those provided include module-specific newsgroups [17] and an anonymous question-asking facility modelled on that used at Kent [5].

Success in the first semester module is a prerequisite for entering the second year of all degree programmes in the School (and thus for the vast majority of the class). The second semester module is a prerequisite for all the single-subject programmes and also for the majority of the joint-honours programmes. Thus failure in either module will probably prohibit a student from entering the second year of the course.

4.1.2 Programming at Kent

The most recent Teaching Quality Assessment rated the teaching in the Computing Laboratory at University of Kent at Canterbury as 'Excellent'. It is a somewhat smaller department (in a rather smaller university) than the School of Computing with about 600 undergraduate FTEs and 40[5] teaching staff FTEs. There is only one single-subject degree programme, *Computer Science*. There are also joint-honours programmes that include computer science but there are far fewer than at Leeds. The content of Computer Science is less theoretical than the course with the same name at Leeds. In emphasis and content the Kent course probably lies somewhere between the Computer Science and Computing courses at Leeds.

The Computer Science degree programme runs for three years with the optional industrial placement year between the second and third years providing the four year option. The programmes are modular with each year consisting of the usual 120 credits. Modules are taught over the whole session with most assessment taking place in April or May. Students are expected to spend about 40 hours a week on their studies (slightly more than at Leeds).

The programming language used is Java. It is taught as part of a module that occupies a quarter of a student's year (in terms of credits at least, since 30 of the 120 come from this course). The weekly formal teaching consists of two lectures and one class (a practical exercise held in a laboratory) each week over the 24 week academic year. The class size is about 180, 130 of whom are reading Computer Science. The single-subject students also study some functional programming in Haskell as part of another module. Success in the programming module is not strictly essential for progression into the second year.

The approach is 'objects first' with simple objects being used in the first assignment. The platform is Windows NT® using what amounts to a simple IDE consisting of an

[5] This is actually higher than the number at Leeds since a higher proportion of the staff at Kent are involved in teaching.

editor with a simple development environment added. All the required software is available free-of-charge and many students install it on their home machines. Early exercises involve altering or extending existing programs and many are graphical (for example using method calls to move a small graphical object around the screen).

Assessment is 20% by coursework (half of which is carried out by an 'examination-style assessment') and 80% by a formal end-of-session examination. In the 2000/2001 session all coursework was carried out in pairs and exercises (many of which had only a small summative assessment value) were submitted fortnightly.

Programming is a fundamental part of the whole degree programme. Students can expect more modules using Java and Haskell and may encounter C and OCCAM. There is no sensible path through the degree that avoids a significant amount of programming.

At present all students take the same route through the module. There are a small number who arrive as highly skilled programmers for whom the programming course offers little and many students with little or no experience who find the programming extremely challenging. The examination-style assessment is carried out towards the end of the first term and is used as the basis for some streaming for the practical classes in the second term. The students who perform better are allocated to groups that may tackle some more complex aspects of programming while those who struggle are grouped together in other classes. The assessment regime remains the same for all.

Staff knowledge of Java is good and there are many more staff involved in teaching the course than at Leeds. Many staff supervise the classes and the atmosphere is informal. Students are free to approach staff for help and many do. Support is also available through bulletin boards and from an anonymous question-asking system presented from a web page (in fact the original system that has been adapted for use at Leeds).

4.1.3 Comparison

The students from the two departments in the study are on the whole comparable. They have been recruited from basically the same pool of applicants and they then go on to follow similar degree programmes. There is no particular reason to believe that the geographical difference in catchment area would make any difference to the ability or learning of the students.

Other work carried out at Leeds and Kent [16] has demonstrated statistically that the cohorts at these two institution can reasonably be treated as a single one for a study such as this and they have been treated as such in similar studies [24]. However, there are some differences which may have an influence on the present study. The main attractions to students at Leeds are said to be the nightlife in the city while students choosing Kent are attracted by the sporting facilities [130]. This shows something of a different motivation for choosing an institution but should have no impact on intellectual ability or ability to learn to write computer programs.

The main difference in the programming module is that Java is taught at Kent as opposed to C++ at Leeds. The computing environment is also different. That at Kent is based on a Microsoft operating system and in this way uses a platform with which the students are much more likely to be familiar before the start of the course. The Unix-based system at Leeds is much more likely to be unfamiliar and thus has a steeper learning curve. This means that students at Leeds are learning to use a new computing environment as well as learning to program. This is something that might be expected to cause them extra difficulties. However, there is scant evidence anywhere in the extensive literature that the language or platform used has much influence on the effectiveness of the learning. Both Java and C++ are high level object-oriented[6] languages and they each have their own peculiarities and foibles.

There are more students in the class at Leeds but this is not particularly important in a lecture setting. Both teaching schemes include smaller group activities with plenty

[6] In the case of C++, just about object-oriented.

of opportunities for students to gain considerable hands-on experience with plenty of immediate support available. The other support mechanisms are identical (the Leeds anonymous question-asking facility is a simple adaptation of the code for the system used at Kent).

The larger class at Leeds appears to be more diverse if only in terms of the number of degree programmes represented. The great majority of the students in the class at Kent are studying a single-subject programme while almost half their counterparts at Leeds are reading a joint-honours degree. The Kent class is thus somewhat more homogeneous in terms of chosen degree programme at least.

It follows that the measures at Leeds to provide extra support to strugglers and extra challenges for experienced programmers have no formal counterpart at Kent. They would be uneconomical (or at best less economical) with the smaller class and the streamed lab sessions may well render them unnecessary. Equally, the need for them has never been seen to arise.

While both departments have modular degree structures that in place at Leeds is rather more compartmentalised with modules being very much free-standing with limited interfaces to others. Modules at Kent are somewhat broader. A particular issue is that programming is a part of other modules at Kent whereas at Leeds it is confined to its own modules. This means that programming is a much more integral part of the degree at Kent and has much closer links with other subjects. This is possible mainly because there is only one programme and no 'softer', less technical, course to correspond with Information Systems at Leeds.

There are other differences. The module at Leeds is currently taught by two staff who have responsibility for all lectures, classes, and assessment. Rather more staff are involved at Kent where one presents the lectures and many more act as class supervisors in the lab sessions. The assessment regimes are very different. Leeds relies on 100% coursework as opposed to 80% formal examination at Kent. The fortnightly exercises (with little summative value) used at Kent have no equivalent at Leeds except perhaps the exercises carried out in the weekly lab sessions (but these have no summative assessment value at all).

Overall it is reasonable to treat the students at Leeds and Kent as a single cohort when a study does not deal with the intricate details of the course that they are following. Their motivations will have been influenced by a range of issues before they started their course and these are quite independent of their chosen institution or of the precise degree course they have chosen. During the course they will have much the same experience with much the same material being covered and assessed in similar ways. A Kent student would certainly recognise the experience of a Leeds student, and vice versa.

While the institutional differences between the two groups of students can be ignored there may remain differences in the other factors that impact on their motivation. A thorough investigation of this would require a highly fine-grained investigation into the activities and cultures of the two departments and is beyond the scope of this study. A brief informal investigation, in the form of an hour spent chatting with some students from Kent, seemed to confirm that the two groups of students were comparable. The students at Kent had undergone experiences and had developed attitudes that would certainly not have been out of place at Leeds.

Even though the two classes will be treated as one throughout the following surveys and discussion, the different responses from the two institutions will no doubt be of interest to some. For one thing they will show whether the decision to treat the two cohorts as one was correct! For this reason the key tables of data in the following sections are reproduced showing responses from Leeds and Kent separately in appendix D and a complete set of results (broken down by degree programme as well as by institution) is included in appendix E.

4.2 Before the Module

The aim of this part of the study is to establish the dominant motivations of the two classes at the earliest possible point and certainly before any serious programming has been started. The students have all made the choice to start on a computing degree

programme and, as part of this, have enrolled for a programming module. Why have they chosen to do this? What is their motivation?

The following sections, and also those for the subsequent surveys, present summaries of the data gleaned from the questionnaires. The raw results may also be of interest and are included in appendix E. The questionnaires themselves can be found in appendix A.

4.2.1 Survey

The first questionnaire (figure A1 in appendix A) was distributed in a lecture at the same point in the course at both institutions. The chosen lecture was the earliest possible in the course before any significant academic content had been covered. In addition to the standard question the students were asked two key free-form questions that addressed two slightly different aspects of their motivation:

- Please write here the **one word** that best describes your reason for taking your degree programme:
- Please write here the **one word** that best describes your reason for taking this programming module:

A total of 365 valid responses were received, consisting of 226 from Leeds and 139 from Kent. The return rate (in terms of the number of students registered for the modules) was around 70%.

4.2.2 Analysis

The words given by the students in answer to the free-form questions were sorted and categorised (using dictionary and thesaurus when necessary) to group synonyms and other combinations of words reflecting similar motivations (there was no attempt at all to prejudge the categories or to work from any predetermined list). The categories that emerged from this process formed the basis for a numerical analysis of the results.

A full list of the words together with the categories to which they were assigned is included in appendix B. Since the questionnaires were anonymous it was not possible to confirm the meanings behind the answers. As many as possible were assigned to categories that seemed to be the most appropriate but the few totally impenetrable responses that remained were assigned to a *don't know* category.

The categories that emerged for the first question ('Why are you taking this degree programme?') were:

- *achievement* – the main motivator is success for its own ends. Typical words were *satisfaction*, *challenge*, and *ambition*.
- *aspiration* – motivations centering on some future goal. Typical words were *career*, *job*, *money*, and (with commendable honesty) *avarice*.
- *enjoyment* – the motivation is found in the enjoyment derived from the process of studying. *Enjoyment*, *fun*, and *stimulating* were typical.
- *learning* – words that indicate that the very act of learning was a motivator. Typically *curiosity*, *enlightenment*, and *learning*.
- *passage* – there was no clear motivation other than studying for a degree being seen as the next step in the stages of education. *Continuation*, *progression*, and *parents*.
- *programme* – the motivation lies in some aspect of the degree programme itself. Typical words were *consolidation*, *relevant*, and *technology*. One had chosen the programme because it was *easy*.
- *university* – the main motivation is the opportunity to go to university and this is seen as a social rather than an academic opportunity. *Beer*, *independence*, and *Leeds* all featured.
- *don't know* – responses that indicated no clear motivation. Examples were *boredom*, *confusion*, and *stupidity*.

A similar process for the question specific to the programming module produced slightly fewer categories. The full list of words and categories is again included in appendix B.

- *career* – the motivation is to acquire a useful skill that will be used in a future career. Typical words were *career*, *job*, and *money*.
- *content* – the motivator is in some or all of the module syllabus. Words used included *Java*, *OOP*, and *skills*.
- *compulsory* – the module is simply part of the degree course that must be negotiated and there has been no element of choice involved. *Compulsory*, *force*, and *required*.
- *enjoyment* – the main motivation relates to enjoyment of the experience of the module. *Exciting*, *fun*, and *enjoyable*.
- *learning* – it is simply the process of learning something new that is the main motivation. Typical words were *curiosity*, *interesting*, and *knowledge*.
- *don't know* – words that seemed to indicate no motivation at all. Some of the answers were somewhat surreal, including *elderberries*. Other answers seemed to indicate that the student had no idea why they were studying programming.

All the responses were classified under these headings to determine the dominant motivations among the students.

4.2.3 Results

The results from the two free-form questions are tabulated in tables 1 and 2 on the next page. The results from the standard question about general motivation are presented in table 3 on page 97. The slight variations in the total numbers for each part of the survey are due to a small number of incorrectly completed questionnaires

or blank answers to some questions. The presence of some rounding in the percentages means that the percentages do not always add up to exactly 100%.

	Frequency	Percentage
achievement	13	3.78
aspiration	141	40.99
enjoyment	20	5.81
learning	123	35.76
passage	5	1.45
programme	29	8.43
university	4	1.16
don't know	9	2.62

Table 1: Motivation for Degree

	Frequency	Percentage
career	23	6.74
content	47	13.78
compulsory	174	51.03
employment	16	4.69
learning	66	19.35
don't know	15	4.40

Table 2: Motivation for Programming

4.2.4 Discussion

Two factors are clearly dominant in the students' motivation for their degree courses (table 1). The aspiration for some future gain (presumably in the form of a financially lucrative career) is the most important factor. This is closely followed by the desire to learn. The closeness of these two values is surprising. Many now assume that students are taking computing degrees largely (even solely) as a route into a lucrative career in the IT industry (a result already found in Leeds [28]) but these results certainly seem to indicate that a significant percentage of students claim to be committed to learning for its own sake.

	Frequency	Percentage
own satisfaction	168	48.98
please family	1	0.29
please teachers	0	0.00
get a good job	164	47.81
just pass	1	0.29
don't know	9	2.62

Table 3: Attitude to Studies

The low figures for *passage* and *university* are also of interest. Universities often try to sell themselves to potential students on the strengths of their surroundings or of the city in which they are located. The relatively small number of references to such contextual factors suggests that this does not appear to be especially important to the students. Nor do many see a degree as simply the next step after school.

These results indicate that the almost half of students are extrinsically motivated (*aspiration*) and that this is the most popular motivation. Slightly fewer students are motivated intrinsically (*learning*) but this seems to be more to do with the process of learning itself rather than with an interest in computing or programming. It is hard to separate this satisfactorily from achievement motivation and it may be that the value derived from learning is more to do with the resulting sense of doing well.

The responses to the second question (table 2 on the previous page) show that the majority of the students perceived the programming module as simply a compulsory part of their degree. This is worrying, especially when taken with other findings at Leeds and Kent that show that students do not expect to study programming [25]. The staff of most university computing departments would regard programming as a fundamental skill that underpins the whole degree and it is disappointing that the students do not seem to think in the same way. There is a clear danger here that teachers will find themselves addressing the 20% or so who are interested in learning to program and will fail to motivate nearly half the class. If departments truly believe that programming is important they must pass this view on to their students.

The low number of students pointing to career aims as a reason to learn programming is surprising and somewhat at odds with the previous work at Leeds in which students identified programming as a vital skill in demand by industry. This may suggest that students are now approaching programming from an ill-informed position in that they do not now have a clear idea of the marketable skills they need to acquire. The responses to the first question certainly seem to indicate that this would be of interest to them and thus would be an important motivator. The answers to these two questions taken together seem to show that many students are interested in a lucrative career in the IT industry but do not plan to work as programmers.

The results of the final question concerning the students' overall attitude to their degree programme (table 3 on the previous page) are striking. Two motivations are dominant as the basic motivations of roughly the same number of students. These represent achievement motivation and extrinsic motivation. The slight dominance of achievement motivation that emerges here is a surprising finding for those teachers used to dealing with tactical or strategic students. The fact that very few students chose the other three motivations suggests that these are effectively absent at least at the start of the course.[7] It will be interesting to see how this motivation develops over the course and, in particular, whether the interest in achievement is maintained.

These were students in the first week of the first semester of their first year. Their attitudes will obviously change as they go through their course and these changes are investigated in the following sections. It is possible that some students will discover a genuine interest in programming or computing and from this will come to develop an intrinsic motivation. When they are better informed some may come to value the learning of programming more for more extrinsic reasons. During this time it is also to be expected that the poverty of student life will lead some to become more interested in the possibilities of future financial gain and less inclined to learn for its own sake. However, it seems that there is very little evidence from any question in this first survey that any significant number of students are motivated by an intrinsic

[7] This also confirms the suspicion that no students are motivated primarily to please their lecturer. This is a great shame.

interest in the subject itself. This is surely a somewhat depressing observation for computing educators.

Students will not learn about computing at all unless they are taught to (or somehow otherwise come to) value the outcomes. While this survey suggests that they are not motivated by an interest in the subject, it does show that they are motivated for other reasons. In a sense it matters not at all what these reasons are. Students will engage and participate in the activities devised for them as long as they are motivated for some reason. It appears that at the outset the value part of the expectancy-value equation is just about present and correct. The implication is that a teacher approaching a first-year programming class cannot afford to assume that the students are motivated to learn to program. Students have a set of aims – an agenda – that means that they are in a programming class. It is not always the case that they want to learn to program or even that they are at all interested in the skill. A teacher must be aware of this motivation and must attempt to address and develop (or at least maintain) it.

This study now moves on to investigate how the students' attitudes and motivation have developed after their first semester's work. The focus remains very much on the value that they attach to success in this work.

4.3 Halfway Through the Module

The halfway point in the programming courses was reached at both institutions after about eleven weeks just before the Christmas break. At Kent this was simply a break in the single programming course but at Leeds it represented the end of the first programming module (although the vast majority of the class would go on to study the second module in the following semester). The material covered to this point was much the same at each institution with the notable exception that the students at Kent had been programming with objects from the outset while those at Leeds had not yet used them. However, both groups of students would have been expected to

have spent about the same amount of time on their programming. The aim of this second survey is to investigate the effect of this first semester's work on the students' motivations. The focus is naturally on any possible changes.

4.3.1 Survey

The survey mechanism from the first part of the study was not entirely satisfactory. The 'one-word answer' approach provided responses that were straightforward to analyse and categorise but there were clearly instances where it was too crude. There were a few questionnaires (no more than a handful) where the students had tried to select more than one option, had written more than one word, or had otherwise managed to convey that their motivation could not be adequately described in a single word.

The basic design of the questionnaire was retained in spite of this. In retrospect a more flexible design would have been better but any dramatic changes would have made comparison with the results from the first questionnaire extremely problematical, if not impossible.

The second questionnaire was once again distributed to the students in a lecture at the two institutions. The lectures were at the same point in the courses – the last possible before the Christmas break. This time the two key free-form questions aimed to look back to the part of the course just finished and at the same time forward to the next semester's work. They were:

- Please write here the **one word** that best describes your attitude today towards the C++ or Java programming course you have done this semester:
- Please write here the **one word** that best describes your attitude today towards the programming course you will take next semester:

The questionnaire also included the standard question about the students' overall attitude to their degree programme. The full questionnaire is included as figure A2 in appendix A.

There were fewer valid responses this time. Lecturers at Leeds and Kent both reported a great deal of absenteeism (something which appears to be an increasing problem and not just a problem in programming classes). There were 262 responses, consisting of 165 from Leeds and 63 from Kent. This still represents just over 50% of the registered students.

4.3.2 Analysis

As before, a great deal of variation was to be expected in the answers to the free-form questions. The same collation process was used as for the first questionnaire. The answers were grouped and synonyms were removed to allow some general classes to emerge. At an early stage in the analysis it became apparent that two very broad categories were appearing. One related to the experience or expectation and another focused more on the difficulty or otherwise of the material itself. Each of these attracted answers at the extremes as well as more neutral views. The four extremes provided final categories as did the more neutral responses. As expected, a *don't know* category was also needed.

The same categories were used for both questions although the interpretation for each is slightly different. The complete lists of words and their categories are included in appendix B. For the first question they were:

- *positive* – the student has in some sense enjoyed the module or has found it interesting. Typical words were *enjoyable*, *happy*, and *rewarding*.
- *negative* – the opposite. This category includes a variety of reasons for the negative experience and a range of strength of feeling. It includes words such as *boring*, *confusing*, *hateful*, and *useless*.
- *easy* – the material in the module had been straightforward. Words such as *intuitive*, *comfortable*, and *logical* were all used.
- *difficult* – again the opposite. The response indicated that the main feeling

about the course related to the difficulty of the content. *Complex*, *taxing*, and *impossible* all featured.

- *neutral* – the programming module had been no particular problem and so the student appeared not to have any particularly strong feelings about it. Words used included *alright*, *compulsory*, and *reasonable*.

- *don't know* – a final category for answers that were blank, surreal, or otherwise largely impenetrable. These included *money*, *templateless*, and *compromising*.

There is possibly some potential overlap between the *positive* and *easy* categories and equally the *negative* and *difficult*. They are kept separate for the moment since it seems too simplistic to argue even in an educational context that an easy experience is of necessity a positive one or that a difficult experience is negative.

The same six categories were used for the responses to the question looking forward to the next semester's work. The words provided were rather different and in this case the categories are interpreted as:

- *positive* – there is some clear element of looking forward or optimism. *Cool*, *hopeful*, and *prepared* were typical. There were also words expressing a positive attitude such as *determined*, *enthusiastic*, and *aggressive*.

- *negative* – the opposite. The same variety of reasons for this was represented by words such as *horrified*, *despair*, and *boring*. The words expressed a range of strength of negative feeling ranging from *uneasy* through to obscenities.

- *easy* – the remainder of the course is expected to be straightforward. There were only three words used – *confident*, *easy*, and *easier*.

- *difficult* – the course is expected to be hard. The words used were *difficult*, *hard*, and *tough*.

- *neutral* – words that seemed to express no particular opinion either way. These included *alright*, *forward*, and (once again) *compulsory*.

- *don't know* – the usual category for the answers that defied classification. This time they included *undecided*, *no idea* (which is actually two words), and *ahhh*.

Once more there is some potential overlap between these categories. In this case some of the *don't know* words could perhaps have been interpreted as *neutral* and the same observation about the possibility of combining pairs of the first four categories still holds.

4.3.3 Results

The results are shown in tables 4, 5, and 6. The slight variation in the total number of responses to each question is as before due to the small number of incorrectly completed questionnaires. For reasons that are unclear this time several students answered only the final question about their attitude to their course as a whole.

	Frequency	Percentage
positive	96	38.55
negative	44	17.67
easy	9	3.61
difficult	42	16.87
neutral	51	20.48
don't know	7	2.81

Table 4: Looking Back

	Frequency	Percentage
positive	107	46.52
negative	62	26.96
easy	4	1.74
difficult	14	6.09
neutral	29	12.61
don't know	14	6.09

Table 5: Looking Forward

	Frequency	Percentage
own satisfaction	146	55.73
please family	4	1.53
please teachers	0	0.00
get a good job	97	37.02
just pass	11	4.20
don't know	4	1.53

Table 6: Attitude to Studies

4.3.4 Discussion

Table 4 on the previous page shows a healthy proportion of the students with a positive attitude to the course they have just finished. This must be taken in the context of a survey of students who had attended a lecture since it is possible that those who were absent would have had different views. Less than half as many had had a negative experience and slightly more than this remain undecided. It might be suggested that these responses may be linked to prior experience and expectations but that cannot be demonstrated with any confidence from the present data.

It has been suggested that positive and easy are aspects of the same experience which might be characterised simply as *satisfactory*. Similarly it is possible to aggregate negative and difficult into an *unsatisfactory* category and to include neutral and don't know under a more general *neither* category. This gives the distribution shown in table 7.

	Frequency	Percentage
satisfactory	105	42.17
unsatisfactory	86	34.54
neither	58	23.29

Table 7: Looking Back (Summary)

This again shows a high proportion of students for whom the experience has been satisfactory but this is still less than half the class. The contrasting view on this is

that over a third of the class have had in some sense an unsatisfactory experience. This may again be linked in some way to previous experience but the proportion in the unsatisfactory category is worryingly high.

With this summary of the students' views of the course just finished in mind, table 5 is intriguing when compared to table 4 (both on page 103). A few more students are looking forward in a positive way than are looking back in a similar way but many more are looking forward negatively (27% as opposed to 18%) than looked back. These two increases are explained by decreases in the easy and difficult categories and by rather more don't know responses. There is clearly a heady mixture of optimism and trepidation in the air!

These two extremes can be examined further. The categories from table 5 can be combined using the same rules that produced table 7 to produce table 8. This table shows whether the students' overall expectation is for a generally satisfactory or unsatisfactory experience. It is now apparent that almost half the students in the survey are reasonably optimistic about the next course while the proportion looking forward with trepidation is still about a third.

	Frequency	Percentage
satisfactory	111	48.26
unsatisfactory	76	33.04
neither	43	18.70

Table 8: Looking Forward (Summary)

These figures would probably make reassuring reading for the teachers preparing for the second semester. It appears that more than two-thirds of the class are at worst neutral about the second part of their programming course. This is also surprising given the interpretation of the students' views of the first part of the course.

Table 6 on the previous page shows once more that the two dominant motivations for the students are personal satisfaction and the job prospects associated with successful completion. The other factors remain insignificant and these responses may in fact

represent little more than noise in the system. It is interesting at this stage to compare these results with the equivalent findings taken before the course began (shown in table 3 on page 97). The change is shown in table 9.

	Change	Change in %age
own satisfaction	22	6.75
please family	3	1.24
please teachers	0	0.00
get a good job	-67	-10.79
just pass	10	3.91
don't know	-5	-1.10

Table 9: Change in Attitude

The large difference in the two sample sizes (some 80 students) means that any direct numerical comparison between the numbers choosing each option would be meaningless. However, in percentage terms a change in the two dominant categories is immediately apparent. The percentage choosing *own satisfaction* (essentially a form of intrinsic motivation) has increased by almost 7% while that for *get a good job* (extrinsic motivation) has decreased by an even more significant 11%. It certainly appears that the experience of the course has made more students determined to do well mainly for their own satisfaction. This is in line with the Middlesex study and seems to confirm the finding that experience of programming makes many students lose their enthusiasm for applying it in a future career.

It might be argued that the extrinsically motivated strategic students who are keen to get a good job are more likely to be absent from a lecture, busily completing some assessment, while the intrinsically motivated (associated with *own satisfaction*) are more likely to be conscientious attenders. However, there is no evidence for this view and the change is so marked that this cannot be the sole cause. It is clear that some students have changed their views.

The numbers choosing the other four categories of motivation remain very small (and the lecturers will be disappointed to see that it is still the case that no student is setting out mainly to please them) but the change in the *just pass* category cannot be allowed to pass without comment. The proportion choosing this has remained small

but the absolute number (this time from a substantially smaller sample) has increased more than ten-fold from one to eleven. It is not unreasonable to suggest that these students have had an experience that has had an effect on their motivation. They have presumably moved from a category suggesting some more positive value component of motivation into this essentially motivation-free category. There are many other factors at work here (and the final question refers to the students' degree programme as a whole) but it must be a concern that the number of students confessing to this largely negative form of motivation has increased to this extent.

This second survey shows that at the halfway point the majority of the class remain motivated to succeed. The reason seems to have shifted slightly to the extent that more emphasis is now being placed on intrinsic motivation. This is pleasing reading for any teacher about to embark on a follow-on course with these students. The value component of the motivation equation is in good health even if the reasons for attaching the value appear to have shifted somewhat.

As regards their programming in particular there are signs that the first semester course has not been a universally satisfactory experience. There is, however, certainly plenty of evidence that it was that for many in the class and that on the whole the students are positive about their forthcoming second semester's work. It remains to be seen whether this will change as they go through that second semester.

4.4 After the Module

The programming courses at Leeds and Kent both finished at the end of the second semester (or term at Kent). This final survey set out to investigate the students' views once the teaching had ended. For some of the students this would be all the programming they would study but for most there would be more programming in future years of their degree programme. The focus of this part of the study remains on any changes that are apparent in the students' views.

By this time the content covered in the courses was essentially the same. The Leeds students had spent the second semester learning about objects and classes and the Kent students had continued to build on their earlier work. The expectation would be that both sets of students had now reached the same level of expertise and had spent roughly the same amount of time on programming.

4.4.1 Survey

The final questionnaire was distributed and collected in the final lecture at the two institutions. There was just a single one-word answer question on this questionnaire. This was intended to generate a straightforward summary of the whole programming experience:

- Please write here the **one word** that best describes your attitude today towards the C++ or Java programming course you have done this year:

To complement the usual question about the students' overall attitude two other questions were added. The first asked the students to choose a statement that best described their attitude to programming from the following list:

- Programming is fine. I can do it.
- Programming is OK. I can get by, but I don't enjoy it.
- I never want to do programming again.

This question attempts to evaluate the overall effect of the experience of the module on the students' motivation. In an ideal situation a teacher would hope that most students would choose the first option and that very few would choose the last (these two options represent the extremes of opinion). The second option was intended to sum up a more neutral attitude that many students seem to have after their programming course and it might be expected that some would choose it instead of the first. Nevertheless, most teachers would still hope that there would be very few

students choosing the final category. The students made their choice by ticking a box.

The final question simply asked the students to indicate whether they could see themselves working as a programmer in their future career. Some previous work at Leeds [28] had shown that at the outset of their course many students saw this as their intended (or at least expected) future career and it remained to be seen whether they still believed this at the end. This question would also serve to verify the key finding from the Middlesex study – the finding that following a programming course tends to make students less inclined to aspire to a career as a programmer.

As might have been expected there were once again fewer valid responses received with few students attending the final lectures. This time there were 208 responses, consisting of 167 from Leeds and only 41 from Kent. However, this is still almost a third of the class.

The questionnaire itself is included as figure A3 in appendix A.

4.4.2 Analysis

The one-word answer free-form question produced the expected variety of responses. The words used closely matched those from the responses to the equivalent questions in the halfway survey and so the same categories (*positive*, *negative*, *easy*, *difficult*, *neutral*, and *don't know*) were used. The answers to the other three questions were simply counted and collated.

4.4.3 Results

The results are shown in tables 10, 11, 12, and 13 (all on the next page). The slight variations in the total number of responses are as usual due to the small number of incorrectly or incompletely completed forms. Curiously this time several students failed to answer the free-form question but answered the other three.[8]

[8] This phenomenon becomes worthy of future study in its own right.

	Frequency	Percentage
positive	58	29.29
negative	36	18.18
easy	4	2.02
difficult	46	23.23
neutral	38	19.19
don't know	16	8.08

Table 10: Looking Back

	Frequency	Percentage
"fine, I can do it"	96	46.15
"OK, but I don't enjoy it"	69	33.17
"never again"	41	19.71
don't know	2	0.96

Table 11: Attitude to Programming

	Frequency	Percentage
yes – would work as a programmer	79	37.98
no – wouldn't work as a programmer	125	60.10
don't know	4	1.92

Table 12: Attitude to Career

	Frequency	Percentage
own satisfaction	101	48.56
please family	6	2.88
please teachers	0	0.00
get a good job	81	38.94
just pass	12	5.77
don't know	8	3.85

Table 13: Attitude to Studies

4.4.4 Discussion

The results shown in tables 10 and 11 seem to show very much the same story. In their reactions shown in table 10 almost half of the class gave a positive or neutral answer and this is reflected in the similar response to the question about programming (table 11) where almost half the class consider programming to be fine. Just under a quarter of the class recorded the difficulty of the course as their first reaction but it remains debatable whether this is a good or bad thing. It is striking that hardly any considered the module first and foremost easy (even those who presumably had prior programming experience). Only 18% gave a wholly negative view of the course. This figure is unsurprisingly close to the 20% who never want to program again and a reasonable speculation would be that many of them are be the same people.

As it turns out it is not that simple. A brief further analysis of the responses of those students who never wanted to program again sheds some light on their reasons (table 14). Their views on the module focus on negative aspects (34%) or on difficulty (46%) (the solitary positive response is an intriguing anomaly). This reveals that 14 of the 36 *negative* views (or 39%) in the class as a whole never wanted to program again. The dominant reason why these students want to avoid programming in the future is the perceived difficulty of the activity.

	Frequency	Percentage
positive	1	2.44
negative	14	34.15
easy	0	0.00
difficult	19	46.34
neutral	3	7.32
don't know	4	9.76

Table 14: Never Again

If the categories are once more combined into broader *satisfactory* and *unsatisfactory* groups a less happy picture emerges (table 15 on the next page). The students are now much more evenly split across these three categories. This is largely due to an

increase in the *difficult* category. This is certainly less reassuring since it seems to be the case that almost 40% of the class are having some sort of unsatisfactory reaction to the course.

	Frequency	Percentage
satisfactory	61	29.81
unsatisfactory	82	39.42
neither	64	30.77

Table 15: Looking Back (Summary)

The responses to the second question (table 11 on page 110) also appear to show that the programming courses have been successful as a learning experience. Almost 80% of the students consider that they can program reasonably well (even if many of them do not enjoy it). This is a very pleasing outcome even it it must be taken in the context of representing 80% of the students who attended the final lectures.

There are some less pleasing outcomes. It appears that over 60% of the class now have no intention of working as programmers in the future (table 12 on page 110). Something has clearly changed since the earlier work at Leeds indicated that at the outset the vast majority saw their future career in programming.

This finding is of course in line with the similar study at Middlesex and, when taken together, these two studies show that the experience of following a programming course changes students' attitudes to programming. The change itself may be due to many influences of which the experience of the programming course is only one – one suggestion would be that the students may have become better informed overall about the range of careers in computing – but the change is marked. Especially noticeable was that not a single student taking the Leeds Information Systems degree or its joint-honours variants (some 50 students) said that they would consider a career as a programmer. This is a particularly interesting finding since it was this exact group whose predecessors had expressed precisely the opposite view in the earlier work. Of course, the experience of working as a professional programmer is very different

to that of studying an introductory programming course but these students already appear to have firm views on the former even if they have experience of only the latter.

As would have been expected, the same two factors (*own satisfaction* and *get a good job*) remain dominant in the students' overall attitude to their course (table 13 on page 110). The relative importance of these factors has also remained roughly the same. The number of students choosing the *just pass* option has increased once more in percentage terms but not significantly. The staff will doubtless be disappointed that once again not a single student is motivated mainly to please their teacher. There remains very little evidence of this or of any form of social motivation.

A healthy proportion (in fact almost half) of the students still claim to be intrinsically motivated. This is another nail in the coffin of the belief that all today's students choose computing solely for the career prospects. However, this extrinsic motivation does indeed still account for almost 40% of the class. These are clearly very much the two dominant motivations, with the social aspects in particular continuing to score very low.

The picture that emerges here is of a group of students who are looking back on their course with very mixed emotions. Some have had a positive experience and are enthusiastic about programming. Others have had a much more trying time and are glad that that part of their academic career is, they hope, over. It is easy to see the early signs of the students determined to avoid programming at all costs in their dissertations. At Leeds these students tend to be in the less technical degree subjects (as is probably to be expected).

As an aside, this finding about the Information Systems programme rekindles to some extent the debate about the desirability of a mathematical background for a student starting a programming course. These are the students who would be expected to have the lowest level of formal mathematical attainment (there is no mathematics entry requirement beyond GCSE). They appear to have formed strong views about programming and they are determined never to do it again. They have achieved the

same entry standards (albeit very probably in very different subjects) as have their peers who chose Computer Science but these latter are clearly much more comfortable with programming. Although far from conclusive this does indeed raise once again the issue of the importance of a mathematical or scientific background. While there is no firm evidence here that mathematical ability has any impact either positively or negatively on programming ability the case for or against is distinctly not proven either way.

The closing news is that at the end of this part of the study almost all the students surveyed still appear to continue to attach some suitable value to success in their course. The nature of this value has changed only slightly since the halfway point (suggesting that the changes took place during the first semester). There is little evidence that many of the students have lost their motivation. It merely seems that its nature has changed. Happily this change continues to be from extrinsic to intrinsic factors.

4.5 Summary

The students at Leeds and Kent have been surveyed at three key times during their introductory programming course. A picture of their motivation has emerged at each stage and it is now possible to combine these into an overall view of the development of (or the changes that have taken place in) their motivation through the course. The focus in this chapter has been firmly on the value that the students attach to success in the course. The good news is that all seems to be well. The vast majority of the students continue to value success for various reasons and the rather negative 'I just want to pass' attitude has attracted very few.

Most of the students in the surveys have now successfully negotiated the first year of their course. There will have been some included in the first survey who have left the course for some reason and these will have had a very slight influence on the results. Also, since the surveys were distributed in lectures, they represent the views only of

those students who attended these lectures. This too may have had some influence. It might be suggested that the students who fail are those who do not attend the lectures but the effect of this is probably slight. It could equally be argued that those who felt that they had already done well and passed the course (at the time of the final questionnaire all the summative assessment had been completed at both institutions) would not bother to attend a final lecture[9] but the influence of this is probably also slight. In any case the two possibilities would seem to cancel out each other's influence. There is no reason to believe that the views are not representative.

One question has been used on all three surveys. The students were asked to choose from a list the motivation that most closely matched theirs towards their degree programme. Table 16 shows the responses from the three surveys in sequence.

	Percentage		
	Before	Halfway	After
own satisfaction	48.98	55.73	48.46
please family	0.29	1.53	2.88
please teachers	0.00	0.00	0.00
get a good job	47.81	37.02	38.94
just pass	0.29	4.20	5.77
don't know	2.62	1.53	3.85

Table 16: The Students' Attitude

The most intrinsic form of motivation (achievement motivation – 'I want to do well for my own satisfaction') has remained the most popular choice throughout and is the most popular by some margin at the halfway point. The proportion choosing this increased to over half at this point but returned to something closer to its previous level at the end (so the halfway result may be just something of a blip). The only other motivation to be chosen by a significant number is extrinsic ('I want to do well so that I will get a good job'). The number choosing this had decreased at the halfway point and this change was maintained at the end of the course. These two motivations have consistently accounted for over 85% of the class throughout the study.

[9] I am indebted to Simon Myers for this observation.

The other three choices (it seems reasonable to ignore the *don't know* responses) have attracted few students but in every case the proportion choosing them has slowly increased. The null motivation category ('I just want to pass') in particular has increased from virtually zero at the start to some 6% at the end. This most probably corresponds to students who have become disillusioned with their course and no longer have any positive form of motivation to succeed. A similar comment may apply to the less dramatic increase in the family component of the social motivation category ('I want to do well to please my family') which has also increased. This could represent disillusionment on the part of some of the students but with an emerging determination to persevere so as not to be a disappointment to their families.

It is clear that a significant majority of the class continue to value the outcome of their course. The dominant motivation is, perhaps surprisingly, rather intrinsic (being based on achievement) but that surprise is a pleasant one. The rather smaller number consistently choosing extrinsic motivation will be a surprise to many of those familiar with tactical or strategic students.

The free-form questions used in each survey show a less consistent picture. Some three-quarters of the class were motivated by either *aspiration* or *learning* at the start of the course. At the same time almost half viewed the programming course as simply a compulsory part of their degree. Most computing educators would argue that programming is a fundamental, important, part of the whole discipline. There is perhaps a failure here to convey that to the students.

By the halfway point most were having a satisfactory experience and expected this to continue. Therefore, the programming courses were addressing the needs of about half the class while the other half was having a less productive time. The second survey also showed that there had been a noticeable shift from extrinsic to intrinsic motivation. It might be hoped that this is evidence of an increasing interest in the subject but this can be only conjecture.

At the end of the course fewer students were able to see the course as satisfactory. This change may be caused to some extent by 'end-of-year blues' but it is none-the-less worrying. However, a pleasing number of the class thought that they were reasonable programmers at the end of their courses and so the courses were working on an academic level at least.[10]

This is a key point. It appears that the courses were teaching the majority of the class to program.[11] The estimations of the students are reinforced by the pass rates of both modules (few students failed outright). The problem is that the experience of learning to program is being less than satisfactory (or even enjoyable) for a significant part of the class. They are learning to program but at what cost? A negative experience can surely have a negative impact on a student's *expectancy*.

It is worth emphasising again that this study has confirmed the findings of the similar work at Middlesex University. Students seem to approach a programming course with an interest in learning to program as a route to a career as a programmer. As they learn to program (or as they follow a programming course) and as they learn more about programming this interest wanes. For many it disappears entirely and programming becomes something that must be avoided. The experience of learning to program is indeed powerful.

Finally, although it is not explicitly investigated, there is a worrying trend that emerges as a background theme from this part of the study. Many of the students do not like programming. It is an activity that many of them feel strongly about and they do not like it. A dislike of programming has been shown (unsurprisingly) to have an adverse impact on academic success in programming [57]. It will be interesting to see if this trend is also apparent in the next chapter when the focus is on the experience on a more personal level.

---

[10] This observation does not and cannot take into account the views and experiences of those students who were competent programmers before the course.

[11] Although how many of the students included in this survey could sensibly and practically have been employed as professional programmers at the end of their course is a matter of some concern. At least most had successfully gained a basic understanding.

An apology is in order. The discussion in the chapter is littered with words such as 'probably', 'perhaps', and 'conjecture'. This is a problem inherent in any investigation into so abstract a concept as motivation. It is only ever possible to listen to what the students say and from this to infer their motivation and the reasons behind it. The raw data that form the basis of these discussions and the resulting inferences are included in appendices where they await alternative interpretations.

The inferences presented here seem to be reasonable. The overall picture is of a group of students who remain well motivated throughout their course. This analysis of motivation has been largely in terms of the *value* that they attach to success. In the following chapter the focus turns on to the question of whether they always *expect* this success. If this is not the case then their value becomes effectively worthless.

Chapter 5

Expectancy – The Individual

First Voice: From where you are, you can hear their dreams . . .

This part of the study is more fine-grained and 'close-up' than that described in chapter 4. It works on a more personal level and thus facilitates a concentration on the *expectancy* component of the expectancy-value motivation equation. Expectancy will be strongly influenced by the day-to-day experiences of learning. This point cannot be emphasised too much. If students are to be motivated to learn they must expect to succeed. The definition of success is closely linked with the summative assessment and the students' lowest-level academic need. They must believe that they will pass.

5.1 Methodology

The following sections focus in close-up detail on the experiences of students taking the introductory programming course in the School of Computing at the University of Leeds. Two distinct sets of students are included. The more significant (and

the subject of the finer-grained study) is a group of first year students taking the programming course in the 2000/01 session. The experiences of this group were followed on a weekly basis throughout their first semester course. A shorter survey with a second group consisting of 'veterans' from the previous year's class is also included to place the experiences of the first group in a suitable context

The students all agreed to take part at the start of the study and all gave permission for the publication of the results (suitably anonymised). The students' names have been changed in the sections that follow so as to preserve their anonymity and save them embarrassment.[1] As far as possible their experiences are described in their own words and as they described them at the time.[2] The methodology used for each group is described in more detail in each of the two sections.

The group of veterans is considered first.

5.2 The Year After

The students whose experiences are described in this section are all veterans of the programming courses at Leeds (students who had taken the modules in the previous session (1999/2000)). They were interviewed individually during the first few weeks of their second year when their memory of learning to program was hopefully still fresh. The interviews all followed the same structure and were recorded on the same form (figure C1 in appendix C).

These students had all successfully negotiated both the programming modules in the School of Computing but at different levels of (summative) attainment. They were thus intentionally chosen so that they would represent a range of experience and attainment. The majority had started the programming course with no experience of programming but one who had had significant experience (and in fact followed the

[1] Although the false names that some of them chose are still quite embarrassing.

[2] Only the spelling has been changed, hopefully for the better.

fast-track scheme) was included as a control. The students, who all happily agreed to take part, were drawn from a range of degree programmes, including joint-honours.

Some details of the course that these students followed are necessary as background to what follows. The course was a fairly traditional introduction to C++ programming as might be found in any comparable higher education institution. The approach was 'objects last' with the first semester devoted to a procedural subset of C++. This subset was presented in the traditional order and closely followed a standard textbook [41]. The course started with variables and assignments and then moved through conditional statements, loops, and so on. There was also plentiful coverage of testing and debugging techniques. The second semester introduced objects (classes in C++) and with them some more advanced features of C++. The presentation was by three lectures each week and practical work was supported by a hands-on lab session each week. The students would have been expected to spend about 75 hours on their programming course in each semester.

The courses in the two semesters were administratively separate modules. Each was assessed separately and each was worth one twelfth of the assessment for the year. Success in the first semester was notionally a prerequisite for the second semester but this cannot be enforced until the end of the year. All the students interviewed were required to pass in the first semester for progress into the second year of their course and all the single-subject students needed to pass in the second semester.

The grading system used in the modules is somewhat curious and merits a brief explanation. The final grade is first calculated, as would be expected, on a scale of 0 to 100, where 37 represents a pass mark. This number is then mapped into a scale running from 20 to 90. This has the effect that very high and very low grades are altered. It is impossible to score more than 90 or less than 20.[3]

[3] The theory of this system is that it prevents a single very poor or very good result having a disproportionate impact on a student's overall average. No attempt is made here to justify or defend it!

In the interview the students were asked to reflect on the experience of learning to program and to explain whether they felt they had successfully learned to program. They were asked for their feelings about the ways in which the course was taught and for any words of advice that they would give students on their degree programme starting the programming course this year. Finally they were asked to provide a one-word summary of the experience of learning to program. Figure C1 in appendix C illustrates the structure of the interview.

The following sections are included in no particular order.

5.2.1 Nikki

The Computing course at Leeds was Nikki's second attempt at a degree. She had started a course in European Drama and French at another university but had dropped out. Some work experience using computers had aroused her interest and so she had decided on this somewhat radical change of direction. At the start of the module she considered herself a complete novice. She managed to achieve a final grade of 59 although she did not consider that she was by any means a competent programmer.

Nikki felt that the programming course had been proceeding reasonably but had suddenly and without warning become much harder. She had found this a "scary experience" and felt that she had panicked. Some time reading the textbook seemed to have helped but she had the feeling of being behind for the rest of the module. The learning curve was too steep and was not consistent.

Her biggest problem during the module was (in her own words) "motivation". She felt that programming represented a "lot of effort for little return" and disliked the way that C++ was so "pedantic" about "annoying little details". Nikki's advice would be "don't get left behind by not keeping up to date, and it's fatal to miss a lecture". Her summary was that learning to program had been an "experience". This experience was not a wholly negative one but was certainly far from wholly positive.

Nikki's experience of a sudden change in difficulty could well be evidence of learned helplessness. She missed some basic concept (perhaps not through her own fault) and was thereafter unable to follow the course. This seems to have provoked quite an emotional response about the "annoying" process of programming when she found that a lot of effort was required for a small return (one of the key reasons why learning to program is so difficult). The emotional nature of this reaction also hints that she did not feel totally in control of the situation. This was something that clearly had a powerful effect on her motivation. It is to her credit that she had clearly developed some reasonable coping strategies and had come through the module successfully.

During the first semester of her second year Nikki changed course to Information Systems. The main reason for her change was to avoid the programming modules in the Computing course (she actually struggled with the first one for a few weeks but eventually had to admit defeat). Whether she was capable of completing these courses is uncertain. What is certain is that she believed that she could not.

5.2.2 David

David was a joint-honours student studying Information Systems and Management. He had enjoyed the programming part of his course. He described it as "relevant, practical, hands on, interesting, and satisfying" and added that he had particularly enjoyed the "sense of achievement when the program works". Although he had been a novice at the start of the course he had achieved a very impressive grade of 84 (when the maximum attainable is 90). He summarised the module as "challenging".

David went on to add that he had no intention of studying programming further and certainly did not intend to pursue it as part of a career. He felt that he had spent too much time on it in his first year and that this had been to the neglect of his other subjects. He thought that writing serious programs would be "drudgery" and would bring no new challenges. It was these challenges, rather than the content, that had made the module enjoyable. David's reaction perhaps starts to throw some light on the phenomenon of students losing interest in programming as a career.

The lectures had been effective for learning or at least had been effective as part of an overall learning strategy. David's strategy had been to listen to the lecture and to then go and try out what he had seen in the lab. He had some sort of mental model of the programming process and the workings of the computer and felt that this evolved over the module as he learned more (constructivism at work). The practical part of the module was definitely the most interesting and rewarding. David saw many of his peers who "just wanted to get the answer" and would very soon give up and approach staff as an alternative to working out the program themselves. David preferred to work on the program on his own and admitted that sometimes he was perhaps too stubborn about asking for help. As advice David would tell students to "go at it and go to the lab, but don't get too wrapped up in it and spend too much time – and go to the lectures!".

David approached learning to program with a mature attitude and was successful. He relished a challenge and it is this that made him enjoy the module. It is a little disappointing to find a student who has succeeded in the learning aims of the course but who is at the same time determined to avoid programming in the future.

5.2.3 Kelly

Kelly, like David, had chosen to study a joint-honours degree in Information Systems and Management Studies. She expressed her views on programming in emotional and forthright terms to the extent that her brief summary of the experience has four letters and is not reproducible. Thinking about the programming module gave her "nasty thoughts" and made her feel "sick". She attributed this to her background and the fact that she "didn't have that way of thinking" (perhaps evidence of learned helplessness at work again). She had started the module as a total novice and had achieved a reasonable pass standard with a final grade of 52. She was quick to thank the extra help classes for her success. While she certainly did not want to do any programming ever again, Kelly admitted that she could probably write some basic programs if she were forced.

Kelly had learned most from the workshop hours and the extra classes. She thought that the lectures had been counter-productive for her – "a waste of time, I just got into more and more of a state". The assessment had been "relentless" and had taken far too much effort. She had become more and more worried and had become disillusioned with her degree programme as a whole. The biggest help had been access to a sympathetic expert[4] for one-to-one help at a level she could understand and the relief of the extra classes where she met others who were also struggling. Her advice would be "don't leave it too late – it won't go away – and ask for help when you need it".

Kelly has clearly not had a pleasant experience in her programming course but at least she has survived. She now believes that programming is simply something that she cannot do and she attributes this to something in her background. Kelly had encountered an "educational novelty" [44] and, as predicted by Dijkstra, had not enjoyed it and had been unable to cope.

5.2.4 Mike

Mike had some programming experience from his 'A' level course. This should have been an advantage even though it was in a very different environment (Microsoft Visual Basic®) to that which he would meet in his Computing degree course. After being unsure of what route to take in the module Mike eventually decided to join the fast track after having also attended the standard lectures for the first few weeks while he found his bearings. He was pleased with his final grade of 84 and considered himself a competent C++ programmer.

The impression of the fast track was of "being thrown straight in". C++ was not straightforward to learn and was "unforgiving" of small errors. There was added pressure because failing the module had significant implications. Mike did not enjoy programming the "trivial" tasks of the standard coursework where "everyone was

[4] Modesty forbids.

writing the same, and there's no creativity". Programming would be much more rewarding and more motivating if he were developing a program for himself. This problem was addressed to an extent in the fast track where the assignments were "not as boring". He estimated that he would have needed 10% of the material from the standard lectures and his advice would be to consider very carefully which route to take through the module and then to "buy a really simple book".

After some initial uncertainty while he found the best way for him to learn Mike has had a positive experience. It is interesting that the enjoyment from the module comes to him (as it did partly for David) from the creativity of the programming process rather than anything more directly related to the academic content. Mike's is a clear success story although it is debatable how much of this success was directly due to the programming course itself and how much can be attributed to Mike's previous programming experience. This experience certainly seems to confirm that prior programming experience is a distinct advantage.

5.2.5 Josh

Josh is included in this study as a form of control. He started the Computer Science course after many years of programming various Acorn® home micros and before the course was able to claim extensive experience in BBC BASIC, some machine code, a little C, and some C++. He naturally followed the fast-track route through the module and achieved a final grade of 90 (the maximum possible). Josh would probably not be too offended to be referred to as a 'typical computer nerd'.

The fast track had been a good thing. It had meant that Josh had not been forced to sit through lectures dealing with the basics. He felt that there would have been a strong temptation to miss all the lectures and then miss those rare concepts that would have been new to him. The process of learning C++ was simply a case of "picking up the syntax". While he had been "left to get on with it" there was always support available if he needed it. It was especially good to be working with similar people to himself. He described them as "people on the same wavelength".

Josh considers programming "great fun" and held this view before the start of his course (he claimed to have started programming at the age of six). He "supposed" that he was now a competent C++ programmer. He had still found the programming module a "challenge" and Josh's advice would be "don't leave it all to the last minute" (both these comments must be taken in the context of the demanding fast-track route that Josh had chosen).

Josh's fast-track experience reinforces the description in section 3.1 of the process whereby an experienced programmer acquires a new language. It had simply (and that is an important word – it was simple) been the process of learning a new syntax. Most of this could be done from a textbook and then by experimenting by writing a few programs. Any complications could be resolved with quick access to some guru. Josh had probably gained little from the course that he could not have picked up unaided on his own.

5.2.6 Keera

Keera (another student studying Information Systems and Management Studies) was relieved to reach the end of the programming course. She had started as a complete novice and was relieved to find that she had achieved a final grade of 52. She felt that it was an "achievement to have survived". Her main problem was that programming was a very "different thing to learn" from anything she had experienced before (in fact an educational novelty). There is a lot to cope with and it "has to click". Although she did not consider herself a competent programmer she thought she could probably "have a good shot" and "get quite a long way" if called on to write some C++. Programming had not come easily to her. A major problem was that "you have to believe you can do it" (expectancy) at a time when you are overloaded with many other things (the problems of transition to university).

The large class size had been especially intimidating and something of a shock after school. Keera had at first failed to realise that extra help was available but she made

much use of it when she found out. The fact that the extra help was available in a much smaller group made programming much more "approachable". Keera's advice to students like herself is "don't think of it as a nightmare – stick together, work at it and go to the lectures". She had found learning to program "scary".

Keera seems to have coped better with an educational novelty. She seems to have had a rather more positive and determined attitude than some of her peers and it is this that has pulled her through. Importantly, she comments that students must believe that they *can* succeed if they *are* to succeed – they must expect to pass.

5.2.7 Harri

Harri is one of the single-subject students in these sections. He had chosen the Information Systems degree. His main memory of the programming module was of "hours and hours spent in the lab". He had started the module as a complete novice but had managed to achieve a respectable result largely through determination and sheer hard work (he was disappointed that his final grade of 69 very narrowly missed the 70 first class boundary). At the end of the module Harri felt that he was a competent C++ programmer. He remembered the module as enjoyable "as long as you stay on top of the work". The lectures were well taught and useful but the practical sessions were not long enough. It would have been less stressful if he had had access to his own computer so that he would not have had to spend so much time in the labs. The greatest help was a textbook and access to an expert for occasional assistance. The advice for new students was to "keep up with all the examples given, and don't leave this subject to the last minute (or even two or three days before)".

Harri has had a satisfactory experience. He has succeeded in the academic aims of the course but was still unsure of whether he would want to program in a future career. Nevertheless, he seems to have come through the course in a solid position to take on further programming should he choose to do so. The only criticism that can perhaps be made of his strategy is that he should have been a little more willing to ask for help when he needed it.

5.2.8 Kirsty

Kirsty started the programming module with some experience of computing but none of programming. She achieved a reasonable final result of 50. Her memories of learning to program were of stress and constant deadlines. Her main reaction was that she was glad she no longer had to do it and at the end of her first year she decided to transfer from Computing to Information Systems to avoid more programming modules. Nevertheless, she did consider herself to be a reasonable programmer and now found programming enjoyable.

She had learned most from the practical workshops (mostly, she felt, because the class sizes were much smaller) and rated the lectures as effectively a waste of time. The biggest program with the practical work was getting started and getting used to the Unix system. Advice to students learning to program was "don't panic". Overall, learning to program had been a "stressful" experience.

Kirsty found programming stressful but coped well. It is perhaps a shame that she chose to avoid further programming courses. She might well have enjoyed them and would probably have performed and learned well.

5.2.9 Siân

Siân was another joint-honours student, having chosen to study a combined degree in Accounting and Information Systems. She had done a small amount of programming before (as part of an 'A' level course) but rated herself as a total novice at the start of her degree. Her final grade for programming was a reasonable 57. She had joined the extra classes at the earliest possible opportunity.

Siân's views about programming were set in rather emotive terms. She "hated" the programming module and "pitied anyone who had to do it with her level of aptitude" (perhaps learned helplessness once more). She was adamant that she never wanted to do programming again but admitted that she could probably write some simple programs if she had to. The main reason for her problems was, she thought, that

she had failed to understand the first few lectures of the module and had then been totally lost for the rest (clearly learned helplessness). There was no way she could catch up and the relentless assessment meant that there was no chance of a respite. Beyond pity, Siân's advice was to "go to the lectures and listen, read the book a bit, and go and find help straight away". In a word, learning to program had been "hell".

This summary does not paint a happy picture of Siân's experience but it is probably accurate. For some reason she had missed some early material and was then in a quite hopeless position. The additional classes helped to some extent but there was no way she could ever hope to catch up. It is not especially surprising that she was determined never to do programming again.

5.2.10 The Veterans' Experience

The students in this section have clearly had a range of experiences in their first programming course at university. The only common experience appears to be that they passed (and this is only because those who failed were unavailable for comment). Experiences range from that of Josh which appears to have been highly enjoyable to Kelly's which she can hardly bear to remember. This is a small sample but there is no reason not to believe that these are typical experiences and that each example presented here represents the experience of a group of their peers.

It is noticeable that the language used by some, especially those who struggled, tends to be emotive. Siân "hated it", Keera was proud to have "survived", and Kelly's description of her experience could not be recounted in polite company. These are interesting words to be used about an educational experience. It is hard to imagine the same students using the same words about a course in databases or professional issues. It seems that programming has the power to evoke powerful responses from those who try to learn it.

These students have come from a very wide variety of backgrounds. The diversity is obvious. Josh has been programming since the age of six, Nikki has previously started

and left a degree course in Drama and French, and Mike has come from an IT 'A' level. Nevertheless, they have all managed to meet the entry requirements of their degree programme. They were all judged suitable to start the programming course and were thus all expected to succeed. There is little compelling evidence here that students with no previous experience will always struggle – David had none and has done very well. However, Kirsty, Siân, Keera, and Kelly have indeed all struggled but they have at least passed. David is also a counter-example to the suggestion that joint-honours students will struggle more than their single-subject counterparts. There is no evidence of that here. Mike and Josh confirm the not unexpected notion that students with previous programming experience will perform better in programming courses.

Kelly attributes her struggles to something in her background. For some reason programming was simply something that she could not do. There is no evidence of a similar view from any of the others but some do make the point that programming requires special learning skills. For example Keera says that programming "has to click". This is at odds with the view set out in section 3.9 that aptitude has little impact on final results and indeed the view that aptitude for programming does not actually exist. This may well be a manifestation of a form of learned helplessness where the students seize on a convenient explanation for their failure to learn or their struggles.

All these students were learning to program at a time of transition. Keera made the point that she found it difficult to adjust to learning at university after her experiences of school. There is so much going on in the first semester of the first year that it is difficult to devote as much time to programming as it appears to demand. Siân and Nikki both had the experience of coming to a point in the semester when they no longer understood the course. They adopted different strategies to cope (both of which seem to have been effective) and at least they were in a position to notice and took some effective steps. Failure to do so could well have been academically fatal.

There are clearly some shared experiences. The issue of help seeking is a common theme. David and Harri both suspect that they may have spent too much time struggling alone before seeking for help. Nikki, Keera, Siân, and Kelly all joined the extra classes. This would seem to confirm previous experience [22] where such classes have been seen to be much more popular with the female students and confirms the suggestion that male students tend to prefer to work alone [23]. Another general point is that of the advantage of studying in a group of similar ability. Josh reported this as the best part of the fast track route and Kelly, Siân, and Keera all found comfort in working with other strugglers. The experience in the help classes confirms the findings at Monash [65] of the benefit of informal discussion classes. There are also clearly advantages in working in a more general sense with others of similar ability.

All the students report that they have spent significant amounts of time on their programming. Most of this has naturally been concentrated on the assessment. This appears in most cases to have been well beyond the prescribed 75 hours (which is expected to include all the time devoted to the module including lectures). Moreover, many clearly believe that these demands have been such that their other work has suffered. Words such as 'stress' are common in their descriptions and it is surely significant that the most common theme in their advice is the need for good planning and a prompt start to assessments.

Assessment itself is thus another common theme. There is strong evidence of a general view that assessment outside the fast track is unrelenting and gives no chance of recovery for those who are struggling. This makes it especially important that the strugglers seek help as soon as they realise that they need it and implies that their position is difficult indeed if they fail to realise soon enough that they need it. Mike raises the additional point that the exercises used in the mainstream are, in his view, uninspiring. He makes the point that (in his case at least) students are more likely to be motivated if they are writing a program that interests them. The assessment regime that the students followed was perhaps rather theory X in design. There were many summative assessments with rigidly enforced deadlines. The experiences of these students point to the potential benefits of a more theory Y approach.

There are conflicting views on the effectiveness of the various teaching methods. There is general approval of the practical lab sessions but there is much less agreement on the lectures. David seemed to find the lectures sufficient to learn new concepts (as did Harri) but others found them far from helpful. This is especially true of Kelly who seems to have found them entirely counter-productive. It is apparent that the students who seem to have gained most from the lectures are those for whom the whole experience of the module was more straightforward. It is the strugglers who are being let down (and at times almost intimidated) by the lectures.

The language and platform used are mentioned by many. For Josh the learning of C++ was simply a case of picking up a new syntax – it is safe to assume that his programming and debugging skills were already well developed. Others had to learn much more, including how to 'drive' the computer. The basic practical steps of how to get the program into a file, compile, and run it seem to have troubled many. Kirsty found the intricacies of Unix a problem and both Nikki and Mike found the platform unforgiving and pedantic. This confirms the view that a sackcloth (section 3.2.2) environment is not ideal for novice students, at least in the sense that it can get in the way of the business of learning to program.

On a positive note there is a constant theme of achievement and challenge running through all the students' interviews. Even those who have struggled seem to look back on the experience with some satisfaction, even if this is sometimes mixed with some less positive feelings. This is clearly a significant motivator for David who now feels that he has mastered the challenge. It is odd that he now plans to set programming aside since there remain no new challenges.

David was clearly achievement motivated. This is also a strong motivation for all the other students. Their motivation was, as expected, to do well (to pass) in whatever terms they defined. There is no evidence that any of them ever ceased to *value* the outcome of the module. However, there is plentiful evidence that some of them did at some time not *expect* to succeed, largely because they started to feel that they were not in control. Some of Kelly's, Nikki's, and Siân's comments clearly show this point

of view. It must be assumed that this would sometimes have caused their motivation to drop dramatically and it is to their credit that they have persevered and eventually did succeed. The worry is that there must have been students (it is interesting to speculate on how many) having similar experiences to them who did not persevere and succeed.

None of these students have had a first programming experience from which they will not recover. Josh has enjoyed himself learning a new language. Mike and David have done something that interested them and have gone about it in a way that they found satisfying. Harri has worked hard and got a good result. Nikki, Kirsty, Siân, Kelly, and Keera have all succeeded in passing the module and appear to have done themselves no permanent academic harm.[5]

In conclusion it is interesting to pause and consider how many of these students could genuinely program at the end of the module. They had all passed a programming course but were they programmers? David could certainly program and most of the others were prepared to confess that they could have done if they were forced too. Only a few – perhaps David, Mike, and Josh – could have hoped at this point to work effectively as professional programmers or even to be of any interest to potential employers. This highlights a problem. These students are passing an introductory programming course – they are not learning to program. That will come later (and then only for some).

The focus now moves on to the class of 2000. These students will be following much the same programming course under much the same regime. They will enter the same university and department and will have been recruited against the same standards. It will be interesting to see if they have much the same experience.

[5] The striking gender split in these two lists cannot be passed over. This observation (and the preceding discussion) confirms the pattern noted previously at Leeds [22] that men and women seek help in different ways in programming modules (accounting in part for the predominance of women in the extra help classes). It also hints that men and women tend to have different experiences in a programming module in a more general sense. This should be the focus of further study. This is particularly important in light of the continuing alarming decline in the proportion of female students in cohorts enrolling for computing degrees ([24] and [25]).

5.3 The Novices

This part of the study turns the attention to the personal experiences of a group of students following the introductory programming course in the School of Computing at the University of Leeds as part of their first year. The students all professed themselves to be complete programming novices at the start of the course but their experience with IT as a whole varied. The restriction of this part of the study to novices[6] ensures as far as possible a consistent and definable starting point for all the students.

The students were interviewed individually before the start of the course to ascertain something about their background and expectations. Weekly questionnaires followed through the first semester and the whole process was completed with a final interview just before the start of the second semester. The results from these interviews and questionnaires can now be interpreted in the light of the students' final results in the programming module.

There were three groups of students, representing three of the four single-subject degree programmes in the School of Computing. There was one group from each of Cognitive Science, Computing, and Information Systems. The modules taken in the first year by the students on the last two programmes are the same but those taking Cognitive Science would be taking fewer computing modules (with the rest of their time allocated to modules in philosophy and psychology). The entry requirement for all three degrees is the same in that the same 'A' level grades are required for each and none require 'A' level mathematics or any previous computing experience.

The students were chosen at random within some slight constraints imposed by the School of Computing's tutorial system (in effect this meant only that there would be about five students in each group, that the students would all be standard post-eighteen 'A' level entrants, and that there would be a mix of men and women). No account was taken of previous academic achievement or, specifically, mathematical ability or attainment [16].

[6] About half of the cohort at Leeds claim to be complete programming novices each year.

There were seventeen students in the group at the start of the study and all agreed to take part. However, during the semester the numbers fluctuated somewhat. One of the Computing group dropped out early on (in fact it has to be said that he made no discernible attempt to attend anything), one of the Information Systems group dropped out in the middle of the semester, and one of the Cognitive Scientists hardly attended (although he remained registered throughout). The Cognitive Scientists also gained an extra member (a transfer from a joint-honours degree) in the third week. A further member of the Computing group dropped out over the Christmas vacation but this was after the completion of most of the present study and he kindly agreed to complete the final interview by email. One of the Cognitive Science group left at the start of the second semester in order to return in the next session on the Information Systems programme but this too was after this study was complete. These comings and goings give a final total of fifteen students who were followed throughout their course.

There is both a qualitative and quantitative element to this part of the study. The main part is obviously qualitative, based on what amounts to a series of structured interviews (mostly carried out by means of brief questionnaires). A small quantitative element is introduced by asking the students to forecast their expected final grade on a simple scale in order to gain a crude numerical estimate of their expectation.

The preliminary interview collected some basic demographic data and then asked the students to explain their reasons for choosing their degree programme and the University of Leeds in particular. They were then asked whether they felt they had the skills to become a good programmer (and to define what they believed these skills were) and to say whether they expected that they would find learning to program easy. The interview ended with a one-word summary of their initial feelings about the programming module. The form used to record this interview is included as figure C2 in appendix C.

Each week following this (as a small part of their weekly tutorial) the students were asked to record any particular successes or problems in the programming module

in the previous week. They were also asked to record a one-word summary of the experience of the module, and to estimate their final grade on a simple scale (figure 7). This scale deliberately makes no mention of actual grades because the students were

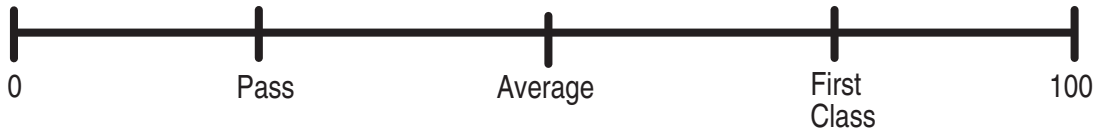


Figure 7: Results Scale

not to be expected to have a precise or common understanding of the complexities of how they would be graded. The scale attempts instead to elicit a more general feeling for how they believed they were performing. In the sections that follow the marks on the scale are mapped to a numeric value for convenience. The 'Pass' line is taken to correspond to the actual pass mark of 37, the 'First Class' to the lower boundary of that class (70), and the 'Average' to a forecast of the mean grade of the module (56[7]). The sections between these fixed points are mapped to grades using a straightforward interpolation.[8] A plot of the resulting grades on a simple line chart gives a rough representation of the development of each student's expectation and morale as they went through the course.

Most students responded each week but some gaps were caused by illness or other absences. A 'half-term' break in teaching around week seven caused something of a hiatus in the responses from the Information Systems and Computing groups when they missed their weekly tutorial. A linear change in the numeric forecast has been assumed where there are gaps in a student's responses (it is clear from the charts where this has happened). The form used for the weekly reports is in figure C3 in appendix C.

[7] The mean grade in the 2000/01 session was in fact 58.

[8] The complication of the scaling of grades from a 0...100 scale to a 20...90 scale was ignored when interpreting the students' responses.

The final interview was conducted just before the final module results were made public and summarised the students' experience. They were asked about their feelings about programming in general, the programming course in particular, and to look forward to the following semester's programming. The interview was again ended with a one-word summary of the course. The form used to record this interview is available as figure C4 in appendix C.

These activities were deliberately kept very separate from the students' normal weekly tutorial (during which the programming modules were often the topic of much heated discussion). Their responses were filed, unread, immediately they were completed and were not to be examined until the end of the semester. The original intention of this was to avoid their providing answers that fitted in with their group's view and to avoid the common phenomenon in such studies of the subjects responding with the views that they think the interviewer wants to hear [118]. When the forms were finally examined it became apparent that several students were recording views that were at odds with, or at least were not expressed as strongly as, those that they expressed more openly in the tutorial. The privacy of the weekly reports was maintained by ensuring that the forms were completed by the students themselves.

The course that these students would follow was in most respects the same as that experienced by the veterans in the previous year (described in the previous sections). The only change of any significance was that a substantial amount of material on testing was brought nearer to the start of the module. The intention of this was to provide a more gentle introduction to the course by leaving hands-on programming until slightly later. For example the first assessment required the students to test a provided executable program and did not require them to undertake any programming at all. Apart from this the presentation, organisation, assessment, and progression requirements were unchanged.

This leaves a C++ programming course still arranged on fairly traditional lines (with an amount on testing as an introduction). The textbook was in a new edition [42] (and some students preferred an alternative [106]) but the order was unchanged.

The topics for the first few weeks were variables, assignments, and so on, followed by conditional statements and the various types of loop. Built-in and user-defined functions appeared around week seven and the course concluded having covered a fairly comprehensive procedural subset of C++.

A few more details of the assessment regime followed in the module will no doubt help in understanding the descriptions of the novices' experiences that follow. There were four pieces of assessable coursework. After the first, on testing, the other three involved practical programming activities. The specification for each assessment was presented as a number of 'levels'. The first level was a basic introduction to the task and the second and third represented what an average student should achieve. The fourth was intended only for the more competent or experienced but was often attempted by the less able as part of their insatiable quest for marks. Each level carried with it a different number of marks. Successful completion of the first two levels was a pass standard, the third level was sufficient for 'first class' marks, and a few bonus marks were available for the final level. There were also two tests carried out in lecture times. These represented 25% of the weight of the assessment, with the rest coming from the practical work. It was necessary to achieve a pass standard in both practical work and tests separately in order to pass the course overall.

The final grading of the module required, as before, a grade of 37 for pass standard, with the raw marks conflated onto a scale running from 20 to 90. The final process of moderating the assessment scaled the raw marks downwards slightly to compensate for the fact that most students had scored very high marks on the assessments.

The initial enrolment on the module was over 300 students. 266 lasted the course and of these 62 secured a first class result. 24 failed.

The following sections do not attempt to present a strictly sequential narrative of each student's experience but rather an overview. They are presented in no particular order.

5.3.1 Lenny

Lenny described his Cognitive Science degree as "inspiring". He had a mainly arts background at 'A' level and had also taken an Advanced GNVQ. His results were good. Lenny's view of a programmer was someone who "likes sitting in front of a screen for hours" and he was sure that this was definitely not him. He did not expect learning to program to be easy but expected to be "fair [or] competent" at it. At the start of the course he was "scared".

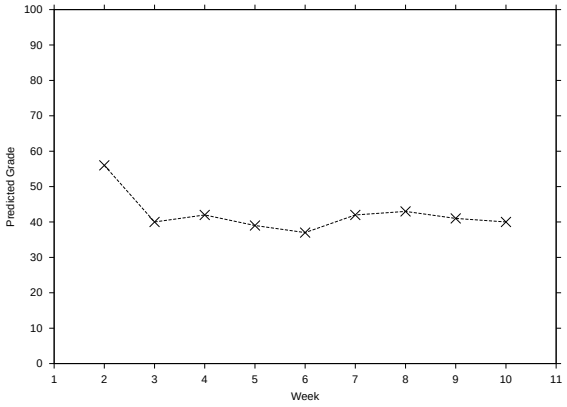


Figure 8: Lenny's Predicted Grades

Lenny's prediction of his grades (figure 8) started with a figure exactly at the mean but then tailed away. He described the first week as "painful" and added that he had "not much confidence at the moment". The following weeks were no better – "help", "confused", "nightmare!", and "clueless". In week four he recorded that he did not "get the variable thing", which seems scarcely a sound basis on which to build a knowledge of C++. By the fifth week he was "turning up to lectures in body but not mind". Things improved after this low point to be "bearable" (this corresponds with the slight upturn in Lenny's predictions). At the end of the module his final grade of 56 was exactly his original prediction.

Lenny described the course as a "struggle" and was adamant that he was by no means a competent programmer. He felt that the lectures had moved too quickly and that he had become lost due to not understanding the basic material (the symptoms of

learned helplessness). The biggest problem was being left to work alone on reasonably complex programs and the greatest help was when he was able to get help on a one-to-one basis. He thought he might just pass in the second semester but was far from sure.

It is worth noting that the story shown by Lenny's reports during the module is rather at odds with the public face he was showing in tutorials. He did indeed seem to be finding the course difficult but he seemed to be coping reasonably well. He certainly did not appear to be having a nightmare. Presumably he found it difficult to articulate his problems, or was unwilling to do so, in front of a group all of whom seemed to him to be coping better than he was. Even so, he succeeded in passing at a level that should have pleased him. However, it is clear that his expectancy of his chances of doing so was variable.

5.3.2 Jackie

Jackie had originally intended to study Psychology at university, but changed to Cognitive Science because it "included a broader range of topics" and "would be more useful for getting a job". She was one of the few to mention this extrinsic motivation at this early stage. She chose Leeds as a university that was quite close to home but still far enough away. Before the programming course she had little idea of the skills that would be required but was doubtful that she possessed them. She was certain that learning to program would not be easy – "I don't expect to be good at it because I have never done anything like it before". In a word she was "scared".

Jackie found the course difficult from the outset. She found that she was able to complete only the most basic parts of the assessments and her predictions of her final grade fell away (figure 9 on the next page). Week four was "horrendous" and the following week was "atrocious" when even the most basic concepts proved impossible to grasp. There is a clear sense that she felt that she was losing control of her progress. Nevertheless, her persistent efforts meant that she was still performing well enough

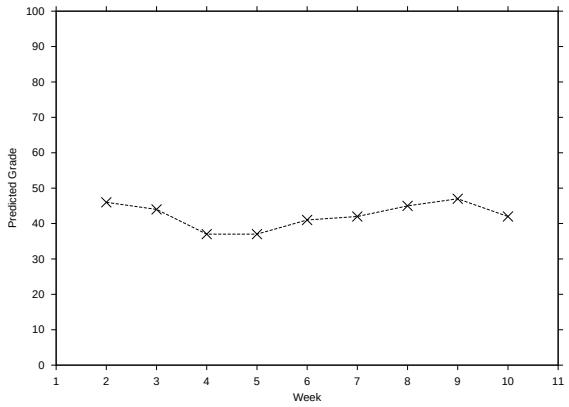


Figure 9: Jackie's Predicted Grades

to pass and she was especially pleased with her result in the first test. Her predictions after this gradually increased (perhaps a beneficial effect of summative assessment on her motivation).

Jackie continually found the whole process of programming "frustrating" and reported finding the intricacies of getting her program into the computer as difficult as those of writing it in the first place. This hints at the problems of mastering a multi-levelled skill. It was very difficult to know where to start with each assignment and weeks when a new one was to be started were "terrible".

Her final result of 60 was something of a surprise for her but a fair reflection at least of the effort that she had put in. It was certainly well above anything Jackie herself had forecast. She was simply "thankful to have passed". Programming was "difficult and frustrating" and she had only "a vague understanding" of the basics. The understanding that she had managed to acquire had been gained largely from the lab sessions because she had followed very little in the lectures. A downside at the end of the semester was a poor performance in her psychology modules. This was something that she blamed squarely on the amount of time she had spent on her programming.

Jackie's experience had not been a pleasant one and is something that she describes in often emotive terms. Her main problem seems to be that she failed to grasp the

basic concepts (as did Lenny) and thereafter had nothing to build on when she was faced with more and more complex assignments. The difficulties were made even worse by her lack of understanding of the mechanics of making her programs compile and run. She was understandably thankful that the course was over.

5.3.3 Will

Will had chosen to attend a university close to his home. He lived at home (some 30 miles away) and commuted for the first few weeks but found this too much of a strain. The main problem was evening access to the labs for his practical work. Information Systems seemed a logical progression from Will's 'A' level in IT, a subject in which he had excelled. Will thought that he had the skills to be a good programmer and expected to be good at it. However, he was prepared to admit that he expected that some parts of learning to program would be difficult.

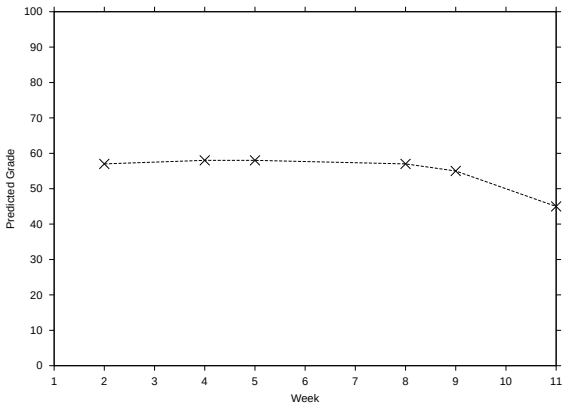


Figure 10: Will's Predicted Grades

Given this attitude it is surprising that Will's original forecast of his result was only slightly above the class average. The first few weeks of the course went well with good results being achieved in the coursework. Will began to describe the work as "challenging" towards the middle of the semester but still had no real problems. This changed quite suddenly in week nine which Will described as "frustrating". This was due to the introduction of overloaded functions, a concept that Will found

very difficult to grasp. Will's expectation of his result from this point on decreased (figure 10 on the previous page) as he encountered more advanced topics.

Will was quite relieved at the end of the module – "it was difficult but I passed". He felt that he "was not by any means" a competent programmer and he had found it "very boring". Overall, programming was "definitely something I don't intend to pursue". His advice to other students would be to "attend all the lectures and lab sessions" and to "start the coursework early".

It is clear that, while Will had passed the module (with a high grade – 64 – well above any of his forecasts), his initial expectations have not been met. The initial picture is of an interested student who is expecting to learn something and to be good at it. By the end of the module he is disillusioned and determined to avoid any more programming. This appears to be quite a disappointing outcome even if the module itself has been a success in purely academic terms.

5.3.4 James

James transferred into the Cognitive Science degree during the second week of the semester, having originally chosen a joint degree in Computing and Philosophy. His progress through the course was steady and his predictions of his grades (figure 11 on the next page) show high expectations and a steady increase in his confidence. His initial predictions were just below the first class border but he became more optimistic as his confidence increased. His final grade of 68 was a fine achievement especially given his late start.

James worked steadily through the course without encountering any real problems. His late start did not seem to be any significant handicap at any stage. Most weeks were "manageable", "normal", or "good" and he was even prepared to confess to being "interested". There was a slight problem when functions were introduced and the course became "hectic" but even then it seemed "pretty mathematical (i.e. doable)". This was only a brief problem because the next week "it's all starting to come

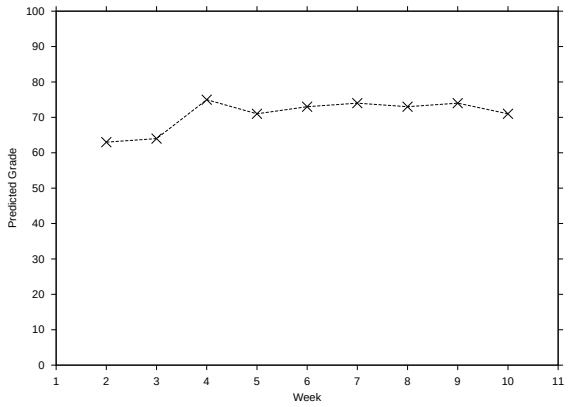


Figure 11: James's Predicted Grades

together". There was still some stress caused by the assessments (the final one was "frustrating") but James progressed well. There was never any question of whether he would pass. The only issue was merely how well he would do.

James's story is one of success and as such is a contrast to the experiences of many of his peers. He has understood the material as it was presented to him and, apart from a few hiccups which he addressed quickly, he has progressed steadily. At the end of the course he felt that it had been "pretty alright" and was "quite easy once you get your bearings". His advice would be to "think like a machine – methodically" (a mental model of a computer). He was looking forward to the next semester's work as more of the same. In the second semester he chose to take the fast-track route. This was quite an achievement for someone who had not programmed at all only four months previously and demonstrated James's high level of confidence.

5.3.5 Steve

Steve took a year out before starting his degree. He chose to study Information Systems because of his interest and for the future career prospects (more extrinsic motivation). He chose Leeds for its reputation and surroundings. He had clear views of the skills required for programming – "clear head, logic, ability to relax" – and was confident that he possessed these skills ("otherwise I wouldn't be here"). He did not

expect that learning to program would be easy but he thought he would be reasonably good at it. He was viewing the programming course with a sense of "anticipation".

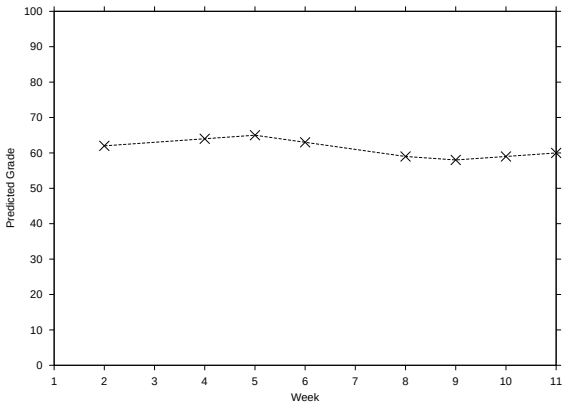


Figure 12: Steve's Predicted Grades

Steve's prediction of his final grade, shown in figure 12, remained reasonably constant consistently about halfway between the average and first class throughout the module. This reflects his experience. Steve initially found that the course was "comfortable" with testing and the basics of programming both "managed well". It became less comfortable in week six when the coursework became somewhat more difficult and was more difficult to start but the overall impression is of steady progress throughout.

At the end of the course Steve "could see the use [of programming]" but had yet to call on it. He thought he was a competent C++ programmer and was satisfied with his final grade (even if at 55 it was slightly lower than his expectation). His comments on the course itself focused on the teaching style and on the organisation of the course. He felt that it could have been pitched at a higher level (something with which many of his colleagues would certainly have disagreed).

Steve's experience was a good one and one that should have pleased him. He had achieved a reasonable level of competence and had done so mainly on his own with the aid of the lectures and course notes. The course had been "complicated" but he had coped well and with maturity. He expected to succeed in the next semester's programming but confessed that he was "nervous".

Steve was a year older than most of the other students taking the module and this appears to have had an impact. He coped well with a challenging course. It seems reasonable to suppose that he was facing fewer challenges in adjusting to university life. His experience was entirely satisfactory and reasonably free of stress.

5.3.6 Karen

Karen's original applications to university were to study Accounting. She decided to change to Computing (another of her 'A' level subjects) and secured a place at Leeds in the summer through Clearing.[9] Leeds was chosen because of its reputation and because it was close to home (although Karen decided to live away from home throughout). Karen did not expect to be good at programming and had very little idea of what would be required. She summarised her feelings at the start of the module as "uncertain".

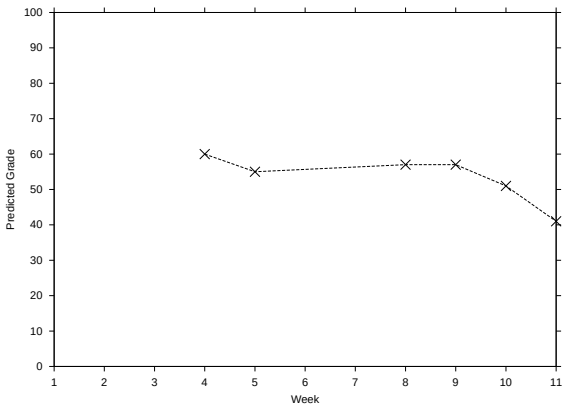


Figure 13: Karen's Predicted Grades

Karen's initial experiences with the programming were positive and she was able to cope well with the material on program testing. Unfortunately, in the early weeks of the semester she was ill (corresponding with the blanks in her grade predictions)

[9] Clearing is a process that takes place in the summer in the UK to allow students still requiring places at university to make late applications.

and missed several lectures. These lectures happened to be those that introduced the basics of C++. From this point on her assessment of her likely final performance decreased (figure 13 on the previous page) from her early above-average expectation. She found that week five was especially "stressful" as she struggled to cope with the lectures and the coursework. From here on the module was "hard" or "difficult" and all Karen's comments focused on the difficulties of completing the coursework.

In the circumstances, Karen's final result of 49 was creditable. After the module she thought it had been "hard and boring". The biggest problem had been the completion of the coursework. Nevertheless, she felt that she was a competent programmer. The biggest help had been the discussions of the coursework in her weekly tutorial. Her advice to other students would be to "start the coursework early and listen in lectures". She was expecting the following module to be more of the same and "difficult".

It is hard to say how much her illness affected Karen's result in the module (and, more importantly, her learning of programming). She was certainly ill at a crucial moment in the module (just as the emphasis changed from testing to development). There seemed to be no way in which she could catch up as more and more new material was presented (especially as she was having to catch up in five other modules at the same time). However, there is no particular evidence that she was experiencing learned helplessness. It seems more that the module simply became a continual struggle to complete the coursework.

Karen's final attitude to programming is clear. It is hard and she was not looking forward to studying it further. This was why she chose to transfer to the Information Systems degree in order to avoid the second year programming courses.

5.3.7 Michelle

Michelle chose Cognitive Science as an extension of her interest in psychology. She had a pure science 'A' level background and results that were exactly the entry

requirement. Leeds was her preferred university because of the social life. At the start of the programming course Michelle expected that learning to program would be difficult but also expected to cope reasonably well because of her "logical mind". Nevertheless, she felt "apprehensive".

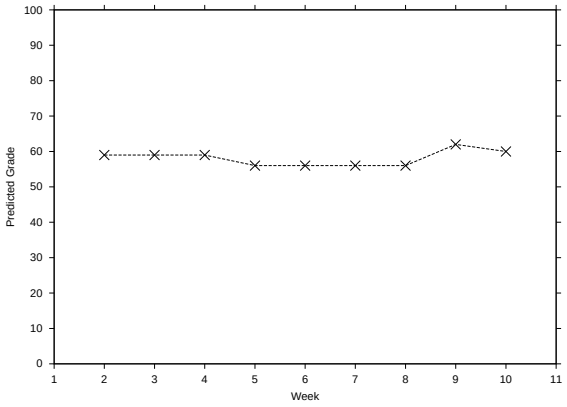


Figure 14: Michelle's Predicted Grades

Michelle's predictions of her grades are quite consistent (figure 14) but there is a noticeable increase toward the end. She made steady progress through the first few weeks (even following James and describing one week as "interesting") but there were some problems when functions were covered in week four and a coursework using them had to be started. This was "terrible" and corresponds with a slight dip in her predictions. However, by the next week the course was "interesting" again and the coursework was completed. All was well until the last week when the final coursework was "horrible" but this too was completed well and Michelle carried on her way to an impressive final grade of 66.

At the end of the course Michelle described programming as "easy now, but wasn't at the time". She thought that she was probably a competent programmer "in some respects" but would benefit from more practice. The most useful part of the teaching had been the practical sessions in the laboratory and the practical demonstrations in lectures.

Michelle's experience of the module was, in her own words, "varied" but overall it was a success. Each new coursework presented new challenges but she coped well and overcame them. Her advice would be to "expect the worst" in the hope that "it might get better". At the end of the first semester Michelle decided to leave the university temporarily in order to return the next session to study Information Systems. She had enjoyed the computing element of her chosen course (and especially the programming) and wanted to study it as a main subject – a success story.[10]

5.3.8 Cynthia

Cynthia chose to study Cognitive Science. Her reasons for choosing this degree did not, perhaps, bode well for success in the computing part – "computers are essential in this day and age, but too boring to study on their own". Her original choice of degree course was English Literature. She had clear views on the skills required of a good programmer ("patience, practice, calm") and was equally sure that she did not possess them ("I'm very impatient and easily flustered"). She was unsurprisingly "worried" at the prospect of starting the course but still hoped to succeed "if I practise enough". On the whole the omens were not good.

Cynthia's predictions of her grade decrease steadily (figure 15 on the next page). Even at the start she was "terrified of failing the courseworks". She also recorded that she "had been given the impression" that her lack of previous knowledge in some sense "limited" what she might achieve. In the third week she was "scared" and thought she would fail. The week after she was in despair because of a perceived injustice in the assessment of her work ("my effort was pointless").

Matters did not quickly improve. The fifth week was "frightening" with the comment "feel I am sinking ... hope I get through this". Week six paints a happier picture with the latest coursework complete – "amazingly, I understand my completed program

[10] A postscript to this is that Michelle returned in the 2001/02 session and successfully completed the first year of the Information Systems degree. Ironically her programming grade was slightly lower.

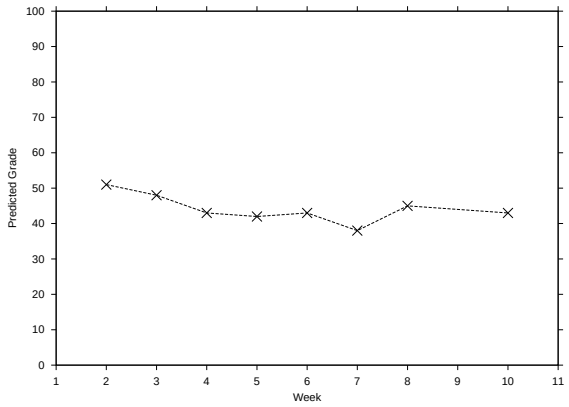


Figure 15: Cynthia's Predicted Grades

and have learned lots" – and an attempt to "get more hopeful (although I probably shouldn't)". Cynthia managed to miss the test in week seven and once more despair set in (with the slight dip recorded in figure 15). After the validation test problem had been solved (or at least the effects ameliorated) with a resit Cynthia started to become more cheerful ("understood the lectures pretty well") even though "I'm sure the coursework will destroy me soon enough". The final assessments were in fact negotiated with only limited suffering ("hectic").

Cynthia's comment at the end of the module were, not surprisingly, rather negative. The course had been "pretty hard and confusing" and she was looking forward to the follow-on course with "dread". She described her problem as having been "thrown in at the deep end". This had meant that "there's nothing you can do but drown". She would not advise others to take the course – "don't do it unless you're desperate to". She did not expect to pass the next module.

In retrospect Cynthia should probably be rather pleased with her final grade (41). This was at least just a pass. She seemed to approach the module with the view that she would find it difficult and would fail (not a good expectation to start with) and almost seemed to be determined to fulfil this prophecy. She undoubtedly worked reasonably hard but could have directed her efforts better. Her occasional brushes with authority and her failure to attend the test certainly did not help. The overall

impression is that she was almost working against the system and would pass (if at all) in spite of the teaching rather than because of it. It is far from certain that she was ever particularly motivated to succeed. Both value and expectancy must be in serious doubt.

5.3.9 Anne-Marie

Anne-Marie's choice of Computing was an extension of her most enjoyable 'A' level subject. As well as being a subject in which she did well, she saw the course as a route into a highly paid job (more extrinsic motivation but this time with obvious evidence of intrinsic motivation). Before the module she had little idea of what skills would be required but "hoped to be good at it". She confessed to being "worried" at the prospect.

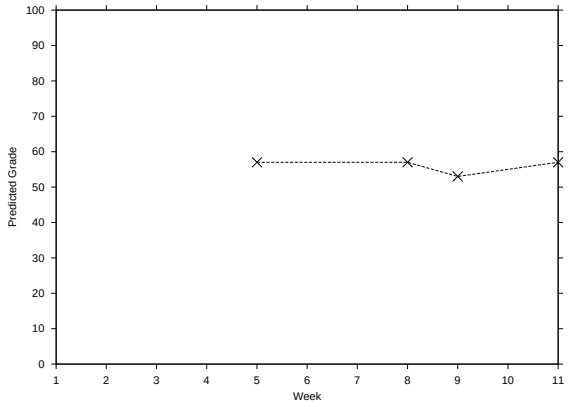


Figure 16: Anne-Marie's Predicted Grades

Anne-Marie's attendance at tutorials during the semester was sporadic as she was frequently absent for (genuine) hospital appointments. Her forecasts of her result (figure 16) show a consistent view of slightly above the average grade and this is in fact exactly what she achieved. Her reports of her weekly experiences are all largely negative and are sometimes expressed in often quite emotive terms – "stressful", "frustrating", "despairing", and "numbing". These reactions all appear to be based mainly on the experience of trying to complete the coursework to her satisfaction.

There is evidence that Anne-Marie drew much satisfaction from her eventual success ("I actually managed to do it!"). By the end of the course she was prepared to admit to "understanding [the course] a bit more".

At the end of the course Anne-Marie summarised the course as "extremely hard and boring". Programming itself was "boring" and Anne-Marie felt that she was "not really" competent. The biggest help to her was making contact with an "expert"[11] who could help her when needed. Her advice to future students was to seek out and adopt such an expert at the earliest opportunity. Looking forward, she experienced "despair" but still hoped to pass.

Although she may not have realised it Anne-Marie made good steady progress through the course and had achieved a solid result (55). She tended to worry when she did not understand the lectures and often sought help immediately afterwards (which is certainly no bad thing). The coursework was always started as soon as it was set and was usually completed in good time. Her (largely misplaced) lack of self-confidence prompted her to seriously consider changing to the Information Systems degree in order to avoid more programming but it seems unlikely that she will do so.[12] Overall, this is a success story in terms of learning if not in terms of Anne-Marie's reaction to the subject.

5.3.10 Ieuan

Ieuan chose his Computing degree because of a long-standing interest in computers. He had not studied computing in his 'A' levels but had achieved good results in Art, Geography, and CDT. Before the programming course started Ieuan did not expect that learning to program would be easy but he expected to succeed "in due time" as a result of hard work. He described himself as "keen" in his first week.

[11] Modesty once again forbids.

[12] As it turned out she did not and in fact managed to complete the second year programming course reasonably well.

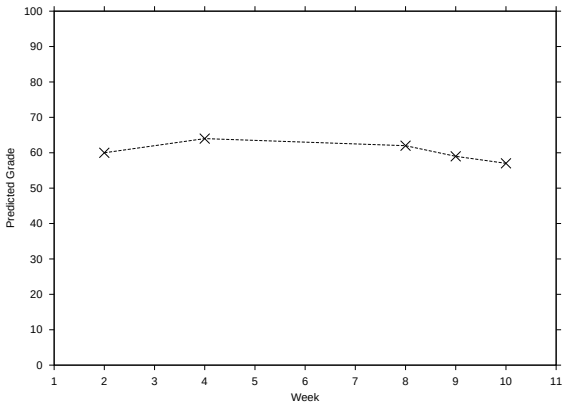


Figure 17: Ieuan's Predicted Grades

Ieuan found the first few weeks of the course straightforward and he consistently reported that it was "going well". This is reflected in his predicted grades, which are consistently above the mean. This confidence seems to have persisted until the end of week eight.

In week nine there is a marked change. The module changes almost overnight from "easy" to "hard". This seems to have corresponded with the covering of functions and function prototypes in the lectures.[13] In the following week Ieuan's predicted grade decreases slightly but still remains above the mean. It is clear that at this stage he was having serious difficulties understanding the module content. The line in figure 17 clearly shows how his predicted grade increases gradually during the first weeks of the course and then tails off after the eighth week as the material became harder. The start of the downward trend corresponds with the introduction of functions and function prototypes.

Ieuan effectively left the course at this point. This was the result of an accident that resulted in a serious back injury which required bed-rest. This period away from his studies produced a large backlog of work in all his modules and this backlog was sufficient to convince Ieuan to leave the course. At the time he felt that the first half

[13] Although other students mentioned this earlier Ieuan's problem was presumably sparked off by starting an assessment.

of the semester had been "reasonably successful" but that this had been undone by his absence. He formally left the university during the Christmas vacation.

Just before he left Ieuan described the programming course as "a good experience which I originally enjoyed, but towards the end of the semester I found out it was not my cup of tea". Programming was "interesting" but "beyond my grasp". Ieuan felt that he could "do a fair bit". The biggest problem with learning to program was simply that he did not enjoy it.

After Christmas Ieuan realised that his decision to leave had been somewhat hasty. His experience in the first semester had demoralised him academically and convinced him that he was not the sort of person who could "expect" to study for a degree with any success. On calmer reflection he realised that this was probably not so and applied to return to Leeds to study a Management Studies course.

Ieuan had indeed had a reasonably successful start to the course. He had successfully negotiated over two thirds of it and was achieving good marks. At this point he seems to have come up against some material that he simply could not grasp (functions and prototypes are notoriously difficult for novices to learn) and was not able to cope. His accident made the situation worse and led to his hasty decision to leave. The main reason for this was probably the need to avoid failure and to retain control of his own academic destiny (there are possibly also elements of learned helplessness too). Had he persevered he would probably have succeeded at a reasonable level.

5.3.11 Carol

Carol chose Leeds as it was one of the few universities at which she could study Cognitive Science. She had no real idea of what was involved in learning to program but was "anxious" at the prospect. She was unsure as to whether or not she would succeed.

Carol's prediction of her final grade (figure 18 on the next page) is remarkable. After an initial estimate exactly on the average level all subsequent weeks were bare passes.

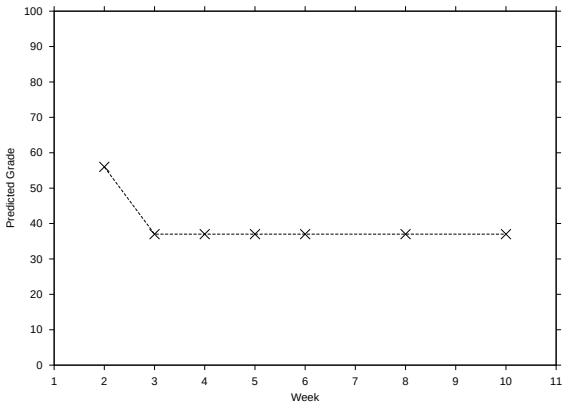


Figure 18: Carol's Predicted Grades

The first week was described as "stressful" with most of the others "terrible". The weeks in which there were coursework deadlines were worse. There was only one slight respite in week eight when the first part of a coursework was completed easily. Her final grade, just below the average at 55, was, by her own admission, more than slightly the result of her seeking out all the possible help available.

After the course Carol was able to reflect on a "terrible experience". She did not consider that she was a competent programmer and was "not very positive" about programming as an activity. She thought that she had passed as a result of the amount of help available and particularly in the lab sessions. Her advice would be "don't miss lectures [or] lab sessions". Looking forward, she hoped that the following course would not be as bad and hoped to succeed but expected to struggle. In a word she was "scared".

Carol certainly worked hard during the module but seemed to discover that the lectures could not be missed (by her own admission she had been absent on more than one occasion but this was much less of a problem in her other subjects). She did not find programming by any means easy but was not afraid to seek help and can attribute her success largely to this. Her predictions reinforce the view [75] that students' expectations should be interpreted in their own terms. Effectively, Carol's aim and expectation were simply to survive and pass and this was achieved.

5.3.12 John

John's original application to the University of Leeds was to study a joint-honours degree in Accounting and Information Systems but he decided to drop the accounting element and entered the School of Computing through Clearing. He chose Leeds "because the campus and facilities looked good" and felt that he had chosen a subject that would be "interesting and useful". John was not sure whether he had the skills to be a successful programmer but was sure that learning to program would not be easy (a "challenge"). At the same time he thought he would not find it too difficult "because I can pick up new things well".

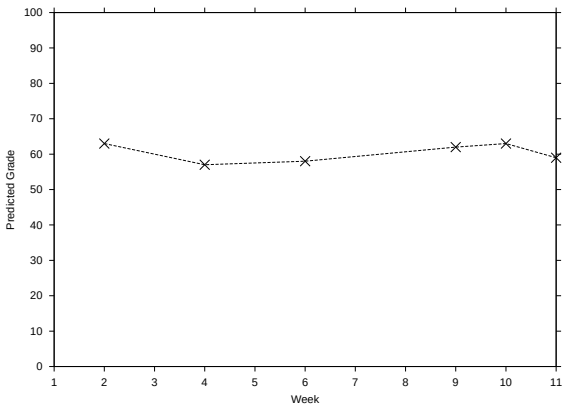


Figure 19: John's Predicted Grades

At first, John found the course "simple" and had no real problems. By the fourth week (when C++ itself was introduced) it had become more "challenging" and his expectation of his final grade decreased slightly (figure 19). From this point on he described the course as variously "hard" or "difficult" but maintained his prediction of his grade in roughly the same area (slightly above the average).

John's final result was actually slightly below the class average at 51. He thought that the module has been "quite hard" and thought programming was "hard and boring" (as did Anne-Marie). The biggest problem had been completing the coursework to an adequate standard. John's advice would be to "ask for help if you have any difficulties". His expectation for the next semester was that the module would be "dull" but he hoped to "scrape a pass".

John was a quiet student who seemed to find it difficult to ask for help when he needed it. It is to his credit that he did so and, indeed, he rated personal help from a tutor as the greatest help to him during the module. He had worked steadily through the module and had achieved a reasonable final result. It is worrying that he was approaching the second semester with a negative attitude of hoping to merely scrape a pass when he was capable of achieving a lot more. It is hard to be certain that he was expecting to succeed even if he did continue to value the outcome of the module to a reasonable extent.

5.3.13 Nigel

Nigel was rather vague about his reasons for choosing Computing. His 'A' level results were exactly the entry requirement but he had not studied computing at any level before. Nigel recorded that his main reason for choosing the University of Leeds was the city's reputation for nightlife. The course itself simply "sounded interesting". He had applied to study a range of degree subjects at various universities – a distinctly odd situation.

Nigel did not expect that learning to program would be easy but expected to be good at it because "I pick things up easily". Before the module he was "curious" about what it would involve.

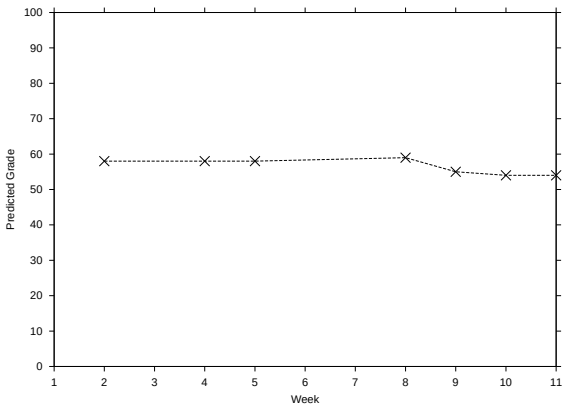


Figure 20: Nigel's Predicted Grades

Nigel's initial forecasts for his grade were slightly above the class average and he was pleased that programming was "not as hard as I thought". This carried on until week five when he suddenly discovered that he had failed the coursework (probably due to having spent too much time on the local nightlife and not enough on his academic work). As time went on he discovered that he had unwisely failed the easiest of the coursework exercises and so had to score even better on the later exercises. At this point his predicted grade fell below the mean (figure 20 on the previous page) and fell away slightly thereafter.

At the end of module Nigel had managed a bare pass (40). This was a considerably worse result than he had forecast and well below his original expectations. He was another that thought programming to be "boring" and "hard". However, he did imagine that he was "just about" a competent programmer. The biggest problem he had encountered was doing enough of the coursework in order to pass and the biggest help had been the practical sessions.

Two other comments made are rather worrying. Nigel recorded his attitude to the following programming course as "hate" and his advice for students following his degree in the next session was "do something else". Both of these are emotive negative reactions (it is hard to see how he would expect to succeed in the subsequent semester with this attitude) but Nigel nevertheless expected to succeed.

Nigel's academic work suffered in the first semester, as it did later (and spectacularly, as it turned out) in the second, because of too great a concentration on the social side of student life (this concentration in fact eventually led to the highlight of his eviction from his hall of residence). He found himself in a position where he had to do coursework of a high standard in order to pass, and probably to a higher standard than he would have done otherwise. This experience had left him disillusioned and rather negative about programming as an activity. It is not surprising that he decided to transfer to the Information Systems degree. His reasoning for this was that he could avoid programming while remaining in Leeds.

In fact Nigel seems to be an example of a form of motivation identified in chapter 4 – his main motivation for going to university was the *social* opportunities that would arise, with the academic content being secondary and almost unimportant.

5.3.14 Max

Max chose the degree in Cognitive Science as he was attracted to its mixture of useful disciplines and its "alternative nature". His 'A' level results were average for the entry requirement but had required a resit. At the start of the programming course Max's views were mostly negative. He did not expect that learning to program would be easy and did not expect to be good at it. He felt "queasy".

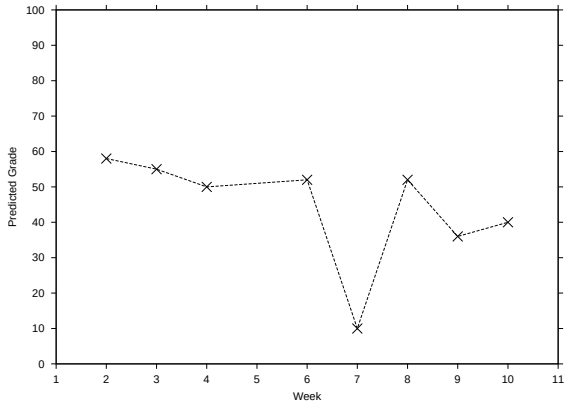


Figure 21: Max's Predicted Grades

Even so Max's initial assessment of his likely achievement was just about above the average. At first he found the course "interesting" and experienced no real problems. However, at the same time his forecast of his final result moved to below average. An explanation provided in week three for this was that "everyone's cleverer than me". As the semester went on Max appeared to become more and more negative about the course. He became "confused" and found the course "hurried". His coursework marks still remained good and were a source of some pride.

Disaster struck in week seven (corresponding to the remarkable dip in Max's forecasts in figure 21). Max managed to miss the first test in the module through oversleeping.

In principle this could have meant that all of his hard-won coursework marks to date would have been lost as well as a zero mark on the test. This provoked the comment "gonna get 0 – gonna fail – gonna die" and the only occasion where any student in the study forecast a clear fail grade. Happily by the following week a resit test had been successfully negotiated, the coursework marks had been secured, and Max was again more positive.

In the following week the final coursework was described as "tortuous" and a marginal fail grade was the forecast. By the time this work had been completed the forecast was at least back in the pass zone.

At the end of the course Max had interesting views about programming – "I detest it – but I really want to like it". The course had been "intimidating" (a reaction which might reasonably be traced back to the missed test or possibly to the view that he was in some sense less clever than all his peers). Max would have preferred more practical teaching rather than lectures and was another to consider the lab sessions to have been the most helpful form of teaching. In spite of his experiences he was positive about the next semester's work, describing his feelings as "hopeful" and saying that he was "approaching [it] with a better attitude".

Max's final grade was a bare pass (at 38). After an uncertain start he had made reasonable progress until the disaster of the missed test. This experience seems to have affected him deeply. He felt that this was very unfair (it was an honest mistake) and should not be allowed to have such a drastic effect on his final result. In essence he saw himself losing control over his destiny in the course. The fact that the missed test could jeopardise his hard-earned coursework marks was especially difficult to accept. Even when this had been resolved it is clear that Max's attitude was much more negative even if he did claim a positive attitude to the following semester. As matters turned out he failed the following semester's module very badly, attending and attempting very little. Partly this was caused by his realising that a pass in programming was not essential for his progress into the second year but there was clearly also an element of believing that programming was simply something he could not do – a clear failure of motivation.

5.3.15 Lizzie

Lizzie chose a degree in Information Systems as a logical follow-on from a subject that she had enjoyed at GCSE and 'A' level. Her 'A' level results were good and included an A grade in Information Technology. Before the programming course she was unsure as to what skills would be required but was sure that learning to program would not be easy. She thought that she would struggle at first because "it will take a bit of getting used to". On the whole, though, she felt "OK" about the prospect.

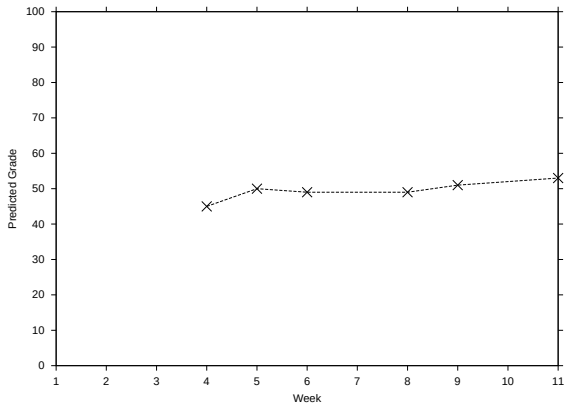


Figure 22: Lizzie's Predicted Grades

Lizzie found the course challenging from the outset but her estimate of her final result increased steadily through the semester (figure 22) even if it was never greater than the average (some confusion over attendance accounts for the missing forecasts from the first four weeks). Her weekly reports all rated the course as "difficult" or "challenging" with all the problems being caused by the pressure of completing the coursework. Her final result of 68 was well above her expectations but was probably a fair reflection of a lot of effort put in through the course.

At the end of the course Lizzie had very strong views about programming. It was (once more) "difficult and boring" and she was convinced that she "didn't like it". She did not consider herself a competent programmer (which is surprising given the grade she achieved) and would tell next session's students that the course was "nasty". Her expectation for the next semester was that the course would be "more difficult" but that she hoped "to pass it at least".

It is difficult to reconcile Lizzie's negative views with the rather more optimistic impression given by her steadily increasing forecast of her grade. She seemed to worry a great deal about her progress and required a lot of help from staff. However, much of this help was simply reassurance or confirmation that what she had done was correct. Lizzie made a very successful start to learning to program even if it seems that she has not quite appreciated that herself.

5.3.16 The Novices' Experience

The most striking thing about this collection of narratives is that so many of the students appear to have had a largely negative experience. Few of them are looking forward to the second semester's programming with any great positive feeling. It is all too easy to see in many of them the signs of final-year student determined to avoid programming at all costs. The positive aspects are that they have all acquired some knowledge of programming (even if they do not all seem to realise how much), have all passed, and have hopefully not done themselves any lasting academic harm.

This is not to suggest that these novice programmers and their experiences are of necessity representative of the entire class. They were chosen because they were complete novices and so their experiences must be typical of those students on the course who had done no programming before. Most students negotiate the course successfully (there were only 24 failures out of 266 students) and many go on to study programming in various forms at higher levels. Nevertheless, the students in the study were selected randomly from among the novices and their experience has been sufficiently consistent to suggest that they are representative of that particular type of student. Other students who had some prior experience (which can be expected to be an advantage [64]) will have had a less negative experience.

Some of the experiences make rather depressing reading. There is the recurring theme of initial enthusiasm being gradually eroded until the course is viewed in a wholly negative light. The language used at this point is strikingly emotive with words

such as “frightening”, “nightmare”, and “horrendous” appearing to carry a weight of meaning beyond their mere dictionary definitions. In the case of Ieuan the experience was a strong contributing factor in his decision to drop out of the whole course.

This initial enthusiasm is generally tempered with no little apprehension. Most of the students are nervous about the prospect of the programming course. None were actually looking forward to it. This is a sobering thought for those teaching a subject so fundamental to the discipline of computing. This points, perhaps, to a problem with the intake or, more likely, to a failure to motivate the intake. Of course the students may well have been equally apprehensive about the other parts of their course. If this is so there is all the more reason to make sure that they are motivated.

This may seem to be an attempt to target the blame for the students' difficulties on the students themselves. It could equally be argued that the course is badly taught or is simply too demanding in terms of time and content. The truth may well be that all these things are partially to blame and that the problem is systemic in nature. Perhaps students who are poorly prepared and apprehensive are being called on to negotiate a course that is simply too demanding.

James, and to a lesser extent Steve, are the exceptions to the general rule. They both seem to have had much more positive experiences (James even chose the fast-track route in the second semester). Both have followed the module in the manner expected and both have achieved creditable results. There is no sign that either of them found the programming module anything other than ‘just another course’. Michelle is another exception, with her enthusiasm leading her to choose to drop out in favour of starting a more mainstream computing course in the next session. But these are three students out of fifteen – only a fifth.

The students' predictions of their final grades give a general impression of how well they that felt they were doing in the course. It is noticeable that the majority showed a decline in the prediction as the semester went on. This is surely a sign of a gradual decrease in enthusiasm and expectation. While students cannot be expected to be able to forecast their grades accurately (particularly in the first semester of the first year) the consistent downward trend is nevertheless surely significant.

Recalling the expectancy-value model of motivation (section 2.1.2), it is clear that many of these students were not expecting to do well (or even to succeed) in the programming course. This attitude was being carried on to the following course – hardly a positive sign. They had this attitude even before the course had started and this is something that again highlights the need to motivate and reassure. In terms of the students' basic needs it is clear that there were many times when they felt that their lowest level need – to pass – was not going to be met.

It is also noticeable that for some reason the programming course (or simply 'learning to program') has a clear reputation. The students have the view that it is difficult (or worse) before the start. Before the course had even started Michelle said that she was "nervous", Cynthia and Anne-Marie were "worried", and Max felt "queasy". While some uncertainty is to be expected it is hard to see this as a positive attitude. From a motivational perspective the *expectancy* component appears already to be worryingly low even at the start of the course.

There is worse to be seen in what follows. There is scant evidence that the students' expectancy increases very far from this low starting point. Their views of the course seem at best to remain constant or, if anything, to become more negative. Only Steve and James appear to have remained confident of success throughout – only two out of fifteen.

The impact of assessment and coursework cannot be ignored. The most obvious point when this came into play was Max's missed test but the majority of other comments relate to assessment and problems completing the assignments or the tests. It is almost as if the students felt driven by the assessment, making it appear to them to *be* the module rather than simply one aspect of it. The assessment regime they were following is not unusual (and is indeed rather less intensive than that used at Kent) but it appears to have had a profound impact on the students' experience and a detrimental effect on their motivation.

Assessment is thus shown to be the most important factor in the novices' experience. It dominates their experience of the module and hides from them the learning on

which it would be hoped that they were concentrating. Cynthia's comment (and the language she uses) is powerful. She was pleased at some success but expected that the next assessment would "destroy me soon enough". Even where assessment is not mentioned it is a background theme. The novices' initial feelings about the course ("queasy", "nervous") surely have more to do with their expectations of successfully negotiating the assessment than with the nature of the subject material.

Perhaps the course that they were following had too much assessment or perhaps the assessment itself was too demanding. But there were only four summative assessments and only three of those involved any actual programming. This would seem to be close to the minimum possible to allow meaningful final summative results to be arrived at. These assessments were all specified to various levels of complexity so it should have been possible for the novices to find an appropriate level. Assessment is necessary in any module but should it really be allowed to dominate to the extent that it appears here that it does?

Closely linked with assessment is the question of workload. The course that these students were following was nominally to occupy them for 75 hours but there is ample evidence that many spent much more. This excessive workload was produced by only four assessable exercises and two tests. The problem seems to be that the students are expending a great deal of effort, are experiencing an immense workload, but are not getting the results that they believe they deserve. It is inevitable that this will have an adverse effect on their motivation.

Overall, though, almost all of the students in the study should be satisfied with their first semester's work even if they do not themselves see that. James, Michelle, and Steve have worked well and have, to varying extents, taken to programming. The others have all worked well and have achieved reasonable results. Apart from Cynthia and Nigel the only real disappointment is Ieuan who could perhaps have persevered.

5.4 Summary

Nothing much appears to have changed. There are many clear parallels between the experiences of the two groups of students. For example David and James were both in some sense 'natural' programmers and both were able to learn effectively from the lectures and classes with occasional advice from an expert. It is worrying that they appear to be such a minority. The others all struggled at times but at least they all succeeded (or would have in Ieuan's case) in the end.

There are a number of instances where the students experienced a sudden change in the difficultly of the material in the module and felt that they were suddenly unable to cope (both Nikki and Ieuan reported this as a major problem). They struggle from this point on, perhaps experiencing some of the symptoms of learned helplessness. A slight variation on this phenomenon is provided by Karen who experienced a similar situation but as a result of classes missed through illness.[14] A course that proceeds relentlessly and quickly is bound to risk this effect.

Nikki's suggestion that the learning curve was inconsistent is connected to this notion of a sudden change in difficultly. This is shown again with this year's class able to cope at first but then becoming suddenly lost. It can be assumed that this is not intentional on the part of the instructors and so presents a clear issue. Experience shows that some topics (parameter passing, functions, arrays) are most likely to cause these discontinuities. It follows that they must be taught with care. Their introduction (and, worse, the teacher's carrying on without verifying the students' understanding of them) can have a profound effect on motivation and expectation.

Some of the veterans (Kelly for example) seem to attribute their struggles to a lack of aptitude. Some of the novices felt that they did have an aptitude, while others did not. There is no compelling reason from these results to believe that aptitude has a significant effect (or even that it exists in any meaningful way). This is in line with

[14] Cynics would, of course, question whether Nikki and Ieuan had actually attended the relevant lectures but there is nothing to suggest that they did not.

other findings at Leeds and Kent which have shown that academic background has little impact on final performance [16] but is at odds with recent findings in Ireland [21] which suggested that a mathematical background was indeed an advantage.

The question of control is a recurring background theme although it is never explicitly mentioned. There are clearly times when the students felt that their own destiny (the *locus of control*) was no longer in their own hands. Max and Cynthia both experienced this in relation to assessment and Nikki almost seemed to believe that the very computer was conspiring against her. The students must feel that they are in control and the educational setting must promote this. It is clear that for many this programming course was taking this crucial control away from them.

A comment made consistently by those among the veterans who had struggled was that an enormous help had been finding others who were similarly struggling (Kelly, Keera, and Siân all reported this as did Josh at the other extreme). The extra help classes that had facilitated this were available to the novices and several chose to join. At the same time several informal pairings emerged (Lizzie and Anne-Marie, Jackie and Carol, Cynthia and Max) who worked together. Such arrangements, whether formal or informal, appear to provide much needed support to the novices and should be encouraged (although not to the extent of encouraging or condoning plagiarism, of course), perhaps by the use of methodologies that promote group working (such as Extreme Programming [157]). Interest in such approaches to programming is increasing [115] and working on programming assignments in a pair has already been shown to promote effective learning [158].

Programming is best learned in a positive social context [152] and this can best be provided in a social group of students at a similar level. This observation also provides strong support for the streaming of the programming class to account for prior experience at as early a point as possible, as is currently practised at Leeds [77] and Southampton [36]. Josh and Mike also reported that this arrangement also has clear benefits for the more experienced programmers.

The work at Leeds and Southampton is based around the idea of dealing with the diversity of the student intake by making available a different, more tailored, learning

experience for certain groups. For example experienced programmers can be expected to learn with little formal teaching while those with no previous experience will need more formal contact hours. The form of teaching is based on a classification [77] of aptitude and experience. It is now possible to devise a similar classification (but this time of *actual* experience rather than *prior* experience) from the present study. Seven distinct groups can be identified (table 17).

Group	Examples	Description
Programmer	Josh	Can already program. Will learn the new language (if necessary) from textbooks with very occasional support from an expert.
Natural	David, James	Takes to programming quickly and easily. Will learn from lectures, experimentation, and textbooks with occasional support from an expert.
Competent	Steve, Michelle	Finds programming difficult but copes well and succeeds. Approaches expert for help when needed and is able to ask sensible questions. Follows most of lectures and can use textbooks for reference.
Worrier	Lizzie, Anne-Marie, Kirsty	Finds programming difficult but perseveres and succeeds in the end. Follows some of the lectures but learns best from practical work. Needs support and encouragement from an expert.
Silent	Lenny, Will, Harri, John	Similar to the *Worrier* but less likely to seek out appropriate support. Will tend to try to solve problems through sheer effort and will probably succeed eventually.
Trier	Keera, Karen	Finds programming very difficult but seeks out appropriate help and eventually succeeds. Needs significant support and advice from an expert, probably as part of a smaller group. Follows little of the mainstream teaching.

Group	Examples	Description
Struggler	Siân, Kelly, Lenny, Carol	Finds programming extremely difficult. Follows none of the mainstream teaching and lacks an understanding of basic concepts. Small group teaching from an expert is essential.

Table 17: Student Experiences

There are clear connections between this new classification and that already in place at Leeds and Southampton. The Leeds classification used four broad groups [78]:

- *Rocket Scientists* – those who can already program.
- *Copers* – those who would find the module challenging but who would cope and eventually pass reasonably well.
- *Strugglers* – those who would find the module difficult and who would not pass without significant extra support.
- *Competents* – those who remain, who will pass with limited support provided when needed.

These can readily be mapped to the student types identified here, as shown in table 18 on the next page. The mapping between the two extreme groups (*Rocket Scientists* and *Strugglers*) is as expected and it follows that the students' experience could to some extent have been forecast from some knowledge of their background. This study has described the experiences that students in the other two classes are likely to have. From this it should be possible to refine the methods currently in place for dealing with the student diversity.

Moreover, the work carried out at Southampton [36] has shown that the students' own description of their feelings before the module is related to their final outcome. This work used a scale of six categories ranging from 'I am an experienced programmer,

Rocket Scientists	$\Longleftrightarrow$	Programmer
Copers	$\Longleftrightarrow$	Worrier, Silent, Trier
Strugglers	$\Longleftrightarrow$	Struggler
Competents	$\Longleftrightarrow$	Natural, Competent

Table 18: Mapping Between Classifications

and will not learn much from this course' to 'I have no experience of computing, and I am worried'. The Southampton work has shown that it is possible to predict the *outcome* for these students. Although its sample size is small, this study has shown that it may also be possible to predict the *experience*. And it is the experience that most strongly influences motivation and expectation.

Is it possible to draw further on this concept of relying on the students' own perception of their abilities to forecast their experience from their feelings before the start of the module? The evidence here is inconclusive but it is interesting enough for it to be worth studying more in the future. Certainly if some basic background information is added it is possible to make some general comments. For example the *Strugglers* were all joint-honours students[15] with a limited mathematical background. The *Worriers* were all single-subject students who had studied some IT in the past. Also of interest is the remarkable fact that all the *Silents* are male and all the *Worriers* are female. This again reinforces the previous work at Leeds and Kent on gender differences in learning to program ([22], [23]). There is clearly something going on.

The question of access to some expert appears in the description of each of the categories and echoes a common comment from the students. Ready access to some sort of sympathetic expert is essential. Most of the students in this study preferred to consult the expert in person (even if for some, like John, this was something of a struggle) but it is equally possible to make such expertise available by electronic means. Several students attributed their success in the module to this help above all other things (a view that probably demeans their own efforts to some extent). With

[15] Here including Cognitive Science.

ever-increasing class sizes it is difficult to see how this can be provided for all students. Perhaps the teacher's personal attention should be focused on those struggling while other students rely on electronic means (which are at least relatively cheap to run) but this seems hardly fair. As well as providing technical help the expert can have a crucial motivational and reassuring rôle, as witnessed by Lizzie's need to be told frequently that her program was correct.

In their different ways Harri and Lizzie show the qualities that are needed to succeed in an introductory programming course – determination and the willingness to put in the hours needed. Others (notably Nigel) seemed to realise that this is what is required only when it was too late. This fits with the concept of learning to program as an educational novelty and something that requires skills and application that are not exercised in previous (or indeed current) academic work. In particular last-minute cramming before an assessment deadline is a recipe for disaster. These demands also do not fit well with the popular perception that the first year of a three-year university course is easy and simply a question of fitting in before the real work starts in the final two years. Some education (and motivation) is needed before the course starts!

The advice given by the veterans was not available to the novices and this is a shame. There are clear indications that they made some of the same mistakes and that they too have come to the same views about the best way to approach a programming course. This should surely become part of induction programmes in all Computing departments. The novices could learn some useful lessons (and meet survivors) before they start the course.

An issue that is specific to the degree programme structure (more accurately to the relationship between the four degree programmes) in the School of Computing, but which is nevertheless interesting, is that many of the students chose to change degree course at the end of their first year. Their main reason (in many cases the only reason) for doing this is to avoid the higher-level programming modules (Kirsty and Nikki took this route and were followed the next year by Karen and Nigel). It may be that these changes reflect a genuine change in interest and perhaps poor initial research

into the degree programmes but the suspicion must be that this is largely a purely tactical move. Again it might be suspected that elements of learned helplessness are at work here or it could just be a case of trying to avoid 'hard' modules. In either case the first programming experience has clearly made a powerful and lasting impression on the students' self-belief and motivation.

The view that the impressions are powerful and lasting is reinforced by the highly emotive language used by the veterans and novices alike to describe their experiences. Negative words such as 'hell' and 'nightmare' suggest that the experience itself is very powerful and is making lasting impressions. It is not difficult to imagine the effect this is having on the students and it is not difficult to imagine them passing this on to the following year's class. Programming thus acquires an unwanted (and hopefully undeserved) reputation and it is a reputation that perpetuates. This reputation incorporates a dislike of programming and it is easy to see how this could lead to a vicious circle including poor performance in programming [57].

This study has confirmed much of what was previously anecdotally believed. The students have a highly varied experience during their first programming course. For many their expectations and motivation are on a roller-coaster as their morale goes up and down due to assessments. It is a strong word but many seem to genuinely *suffer* during the course as they become increasingly anxious and desperate to succeed. There has been little change over the two years covered here and the class of 2001/02 will very probably contain students who have the same range of experiences.

A teacher's rôle in this situation is complex. It is important that academic content is covered as well as possible but there is also a need to pay close attention to the experience that the students are having and closer attention to the effect that this experience is having on their expectations and motivation.

The students' expectation is a problem. Many of them do not expect to succeed. Many of them are not motivated.

Chapter 6

Conclusions

Mrs Pugh: Has Mr Jenkins said his poetry?
Mr Pugh: Yes, dear . . .

The double meaning in the title of this thesis emphasises the two main themes that have underpinned this work:

- The nature of the motivation of programming students.
- A teacher's crucial rôle in motivating those students.

A picture of the motivations of the students has emerged from chapter 4. They are motivated for a range of reasons embracing a variety of both intrinsic and extrinsic factors. Chapter 5 showed how a student's motivation can be influenced by the teacher and the teaching. Sadly the evidence here is that the effect is often negative. The overall conclusion must be that, taking the expectancy-value model of motivation, it appears that the value component is in good health but there are problems with the expectancy.

The objectives of this work were:

- To understand the *experience* of learning to program in today's higher education system in the UK.
- To understand how the students' *motivation* for and *attitudes* towards their programming course alter as the course proceeds.
- To understand the reasons why students choose to take programming courses.

The first of these objectives was addressed in chapter 5 where the focus was on the experience of various individuals as they followed a programming course. The news in that chapter was not, on the whole, especially encouraging. At the very least there are strong indications that the experience is not a pleasant one. Chapter 4 addressed the second objective by investigating the motivations and attitudes of a substantial cohort of students at two institutions and showing how these motivations and attitudes changed as the programming course progressed. The final objective has been more of a background theme. There is evidence that students are not actually choosing to take programming courses. A programming course is simply a compulsory part of a degree programme and is seen as nothing more than an academic hoop through which the students must jump in order to get a computing degree. These objectives are considered further in the sections that follow.

At the same time as presenting the main investigation of motivation this thesis has of necessity considered in passing many other issues surrounding the teaching and learning of programming. This chapter begins with a discussion of what this study has shown about the experience of learning and teaching programming as part of a degree course in the UK's higher education system in the early 21st century. The question of the students' motivation, and of their teachers' crucial rôle in supporting this, is crucial to understanding the students' experience and to making it as profitable for them as possible. After a discussion of the nature of the students' motivation and of the teachers' rôle in ensuring their motivation the chapter ends with some general concluding remarks about the experience of undertaking this study. There are also

some ideas for future work that might be undertaken in this area and some more general reflections.

6.1 Teaching (and Learning) Programming

Teaching programming is a problem. This study has confirmed this and has also confirmed that learning programming is a problem. Chapter 5 showed the range of experiences that a student can have when studying programming. These range from the experience of those students who start the course already competent programmers (classified as *Programmers*), through those who pick programming up quickly and easily (*Naturals*), to those who struggle severely (*Strugglers*). This dramatic range of experience is surely a reflection of the increasing diversity of the student population. Computing continues to expand as an academic subject and expansion inevitably brings more diversity as more and more students are taken from more and more backgrounds.[1] Problems associated with this will worsen in the future.

At any conference on computing education there will be many sessions presented in the area of teaching and learning programming. Papers describe new assignments, new assessments, new languages, new paradigms, and cunning software for detecting plagiarism. Vary rarely, if ever, do the presenters produce any convincing evidence that these innovations improve the students' learning (although some do admittedly make the teacher's lot a little easier). There has to be the suspicion that none of these changes have any real impact. Cynically, perhaps they serve only to make the teachers believe that they are trying *something*. This is thinking and research in Biggs's level 2 terms. It is not getting to the root of the problem.

The following sections consider some of the factors that seem to lie behind the *system* of teaching programming. They are, as always, in no particular order and certainly not in any implied order of importance.

[1] It is noticeable that the student population is more diverse at Leeds than at Kent. This is presumably because of the range of degree programmes available at Leeds.

6.1.1 Diversity

Chapter 5 highlighted the diversity of the students learning to program. It must be remembered that this diversity was within a subset of the cohort since the students were essentially all traditional entrants and novice programmers. This small group had a range of different experiences, learned in a range of different ways, and coped with problems in still more different ways. It follows that the whole cohort must learn to program in so vast a range of ways that it is surely all but impossible to devise a single programming course to suit all the students. It is a shame that in an academic setting with many constraints (finances and physical space to name but two) this is precisely what is required.

Students acquire the skill of programming in different ways. It is quite senseless to attempt to teach them all in the same way. Previous work at Leeds showed a rough-and-ready range of routes through a module [38] based on an idea of something called *aptitude*. This study has shown that there is a diverse range of *experience* that the students have as they follow the single course provided for them. It has also hinted (but for the hint to become a certainty a much more detailed study will be required) that it may be possible to forecast a student's likely experience from information available before the course starts. It must surely be possible to adapt the course (and essentially it is only the course procedures and assessment that need to be changed) to serve the audience more effectively. It is surely more important that the students learn rather than that they all follow the same scheme.

6.1.2 Aptitude

The question of aptitude for programming is indeed a complex one. It is possible to find ample convincing evidence in many studies that there is no such thing as aptitude for programming [101] and experience with aptitude testing at Leeds has been far from convincing. But then logic seems to dictate that there must be some attribute of an individual that makes a student a natural programmer. At the start

of their year the novice programmers in section 5.3 certainly seemed to believe that there was some sort of aptitude and many were very sure that they did not have it.

It is hard to see too much evidence of the existence of aptitude in this study. The students who struggled have little in common and some of the strugglers had much the same educational background as those who excelled. It is worth noting that all the students passed and that even those who had struggled had achieved some level of competence (even if some had competence but lacked confidence). It may have taken them longer to achieve this competence but achieve it they did. The evidence for aptitude remains at best inconclusive.

A particular (and popular) candidate for determining aptitude or likely success (or failure) is a student's previous experience with mathematics. As before, it is possible to find work that shows that mathematical background has no influence on success [16] and work that shows equally convincingly that it does [21]. Confusingly, recent work at Kent [26] appears to once again show some slight connection (this time between success in programming and mathematical *ability* (measured in terms of success in mathematics at university) rather than mathematical *qualifications*). The present study appears at first sight to add some weight to the former view. There is little evidence that those of the students with a more mathematical (or indeed scientific) background found learning to program any easier. The doubt comes from the final survey in chapter 4 which showed a remarkably unanimous view about programming from the Leeds Information Systems students – they did not want to do it. These are students who would not necessarily have had a mathematical background so there may be some influence. Overall, the influence or otherwise of mathematics on programming ability is decidedly 'not proven' as is the case for even the existence or otherwise of aptitude for programming.

6.1.3 Expectation

A diverse student body approaches a degree course for a range of reasons and with a range of motivations. It is clear from chapters 4 and 5 that the potential for a future career is a strong motivator for some but increasingly students choose computing as an extension of the IT that they have studied previously. Herein lies a problem. The IT that they have studied in schools and colleges (in Wales and England probably as part of the National Curriculum) is very far removed from the subject that they encounter at university. Other work involving students studying at Leeds and Kent [24] has shown that many arrive expecting to study a degree that involves learning to use various packages and creating spreadsheets and databases. They are not expecting and so are not prepared for the subject that they actually meet. And worse, they are not prepared for programming in particular. There is a clear need to provide much more detailed information in the schools about the subject of Computing in higher education [25] so that students understand what it is that they are applying to study and arrive more suitably prepared.

Expectation is a two-sided affair. The students have expectations and so do those who teach them. It seems that some staff (and their institutions) have not recognised the dramatic changes that have happened in the nature of the student cohort in the past years and have not adapted to them. It is no longer safe to assume that students approach a programming course with the desire to learn to program – they must be motivated. It is no longer safe to assume that students approach a programming course with the ideal background – they must be motivated. The risk is that many of them will simply regard the programming course as just another compulsory course. If teachers are to maintain the position that programming is such a fundamental part of the discipline that they teach they must be prepared to motivate the students to make them come to believe this too.

There is also an expectation among many teachers that today's student is studying for a degree solely for future financial gain or "to acquire a piece of paper that would make them eligible for certain jobs" [142]. Chapter 4 of this study has thrown serious

doubt on this popular view. It seems that many students are actually motivated by achievement and learning.

There is a problem with expectations from at least two directions.

6.1.4 Control and Learned Helplessness

Students must always feel that they are in control of their own destiny and of their own success or failure. Chapter 5 showed many occasions when this was clearly not the case. Max's missed test and Cynthia's problems with the marking of her coursework are clear cases in point. On both occasions these students no longer felt that they were in control. They felt that their success or failure depended solely on the whim of some other person. They did not think that this was very fair. There is also the sense at times that some of the students appear to believe that they are engaged in some sort of battle with the computer. Its pedantry and refusal to co-operate appear calculated to frustrate rather than to encourage. Perhaps here the students sense control passing from them to a machine.

The question of learned helplessness is closely linked with the question of control. Signs of learned helplessness are common in chapter 5 when students seem to arrive at the view that programming is simply something that they personally cannot do. In one case (Karen) this was largely because of illness but in many others (for example Ieuan, Lenny, and Siân) it seems to have been simply because of the pace of the course. They each failed to follow some basic material and were totally unable and unequipped to cope with what came afterwards. The picture is one of students happily negotiating the course and then suddenly coming upon a mental brick wall that they simply cannot pass – everything on the other side is a total mystery.

While the 'brick wall point' in the module will obviously be different for different students, some of the likely points can be predicted. Experience shows that loops, functions (and especially parameters), and arrays are all likely candidates. A teacher must make sure that the students have understood these topics (perhaps with a simple

formative test) before carrying on. After all, what chance is there of understanding C++ class methods without an understanding of functions and parameters? Or complex data structures without understanding arrays?

The control is clear in a traditional academic setting. Here the teacher is in total command and dictates the pace of the class. This somewhat authoritarian system presumably works when the subject being taught is essentially a body of knowledge. Unfortunately this simple model appears to breaks down when a more complex subject is being learned. For programming to be taught and learned effectively the control must pass to some extent from the teacher to the students. The students must be trusted to decide how they will best learn (theory Y) and must be trusted to engage in the tasks set them. The traditional approach is far too relentless and unforgiving. It leads inexorably to a loss of control and learned helplessness.

6.1.5 Programming in Context

The experience of learning to program at university does not take place in isolation. At the same time as they attempt to come to terms with the intricacies of C++ the students are studying other topics (sometimes in completely different subjects). It is clear that the immediacy of the programming (and in particular the feedback from its assessment) is making many students neglect these other subjects, especially those that will not be assessed until the end of the semester. This cannot be a good thing.

A worrying thought is that this neglect may encourage the teachers of other courses to (in true theory X-style) set more summative assessments to prevent students ignoring the other subject. This potential spiralling in the amount of summative assessment could be disastrous.

The students are also trying to find their feet in a new environment as well as trying to deal with the academic demands of their course. Many are coming to terms with a new city, are making new friends, and are getting used to living away from home for the first time. This is clearly a stressful time of transition and the additional stress from the academic demands cannot be a good thing.

This clearly raises the question of whether it is sensible to retain programming in its traditional place in the first semester of the first year.[2] To move it would be a radical step indeed but there are many powerful arguments in favour of doing just that. As well as the social aspects there are strong academic arguments. Surely it is easier to learn to program when a student has a sound idea of the internal workings of a computer? Chapter 5 also showed that many students are approaching the programming course with some trepidation. Again, surely it would be better to wait until they have settled in. The extra time would also give them a greater chance of becoming comfortable with the university's computing environment. This might reduce some of the problems experienced in getting the program in to the computer.

Section 3.2 raised the possibility of using HTML or JavaScript as part of some more gentle introduction into programming. Given the extremely widespread and persistent nature of the problem in teaching 'traditional' programming, surely this is worth investigating further? A short course in HTML would exercise many of the same skills and would surely be less stressful (not least because HTML is much more likely to be familiar to the students). Such a short course could provide a useful confidence-building introduction to programming. It would exercise skills and processes (the compile-edit-test cycle and the use of editors are two examples) that are not dissimilar to those that will be needed for programming in a more general-purpose language.

In many ways learning to program in an academic setting is an inherently artificial experience. Programming is not something that is learned by listening to lectures or by reading textbooks. Programming is a practical skill. It is best learned by apprenticeship in the company of an expert. The shame is that this is an environment that it is very difficult, if not impossible, to provide in a mass higher education system. There are few, if any, other skills that are taught in this way. For example imagine teaching the skill of driving a car by a series of lectures. No-one would argue that that was sensible so why persist in teaching programming in the same way?

[2] When asked about this, a learned academic of high repute in a prestigious computing school in a prestigious university once said "It has to be taught first. Otherwise the students wouldn't get to use the new computers in the new lab we showed them on the Open Day.".

6.1.6 "Boring and Difficult"

This study has tried to consider some of the key issues surrounding the difficulties of teaching and learning programming in higher education. Section 3.9 considered some of the reasons why the subject itself is difficult to learn and the study itself has highlighted some factors which exacerbate this basic difficulty. The title of this section is a quotation. The most common feeling of the students after their programming course was that programming was "boring and difficult" (or slight variations on this theme). When questioned, they were generally not sure whether it was boring because it was difficult or difficult because it was boring but they were adamant that it was indeed both. It is easy to see how this feeling could lead to a determination to avoid programming in the future.

It is hard to argue that learning to program is not difficult but does it have to be boring? Lectures on programming syntax are certainly unlikely to be particularly inspiring (especially for a student who is struggling) but do assessments have to be boring? A cursory glance through some programming textbooks reveals an extensive range of uninspiring exercises and examples – class grade averages, examples based on libraries and shops, and students and lecturers as examples of objects. Some of the students mentioned that for them an enjoyable part of programming was the creativity. This is something that could be easily be fostered with more open, theory Y, examples and assessments.

The students are correct that learning to program is difficult but that is not to say that it has to be boring. But does it have to be so difficult? Is it perhaps the case that the way in which it is taught (relentlessly, first, with dull examples) makes it appear more difficult than it actually is?

6.1.7 Learning Styles and Activities

Section 3.4 considered the difficulty in identifying the most appropriate learning style for learning to program. It seems that a range of styles is required – a superficial view

might be surface learning for syntax and deep learning for applications – and that a student must therefore engage in activities that promote a range of types of learning. The traditional view of learning styles seems to break down when confronted with a subject as unusual as programming.

The students in this study have certainly adopted a range of learning approaches and have engaged in all manner of activities. Some chose to spend hours in the laboratory poring over their programs while others sought help at the first sign of difficulties (some probably too soon). Many chose to work in pairs, perhaps to double the brain-power or else to share the suffering. It is very hard to see any of them engaging in learning in a particularly deep or surface manner. Nor is it easy to see a particularly serialist or holist approach.

The only model that appears to even come close to describing how a student learns to program is the Kolb Learning Cycle. A student must have some sort of experience by writing a program and must then reflect. This reflection involves adapting the student's mental model of programming in the light of the new experience before passing on to more experiences. This view, which sounds rather like constructivism, is visible in some of the students (David is a fine example) but others seem to have been spending so much time on assessments that they never paused to reflect. It is as if the teaching (or particularly the assessment) is designed to prohibit reflection and learning.

If there is currently an adequate model of the process whereby a student learns to program it is constructivism. Marton and Säljö's deep and surface classification (and others similar to it) does not seem to be directly applicable to such a skills-based area.

6.1.8 Educational Novelty

The term 'educational novelty' is Dijkstra's [44]. He argued that programming was difficult simply because it was beyond the immediate experience of those who learned

it. This means that the study skills that they have used before (and that have served them well) no longer apply. There is ample evidence of that in this study. Many students reported that their strategy for debugging code involved reading a textbook. This is a strategy that might conceivably work but one that is far from ideal.[3]

Herein lies the problem. Programming is such a novelty that the students do not have the learning skills required. However, they have learned many things in the past and have acquired learning strategies that have served them well. They naturally tend to try to apply these. The problem is that they do not map to the new domain and the system breaks down.

Therefore, teachers do need to understand more about the ways in which students learn to program. If students are wrong to try to learn programming in the way that they have learned other subjects in the past teachers are equally wrong to try to teach them in the way in which these other subjects are taught. While programming courses at different institutions are, of course, different in their detail the underlying way in which programming is taught (mirrored in the mountain of programming textbooks) appears to vary little between institutions.

Programming is learned by *experience* and *reflection*. It is an important part of the teacher's job to facilitate these, not just to give lectures on syntax.

6.1.9 Summary

There are many problems and issues surrounding the teaching of programming, of which the key theme of this work – *motivation* – is only one. A cursory glance at the submissions to any conference on computing education shows many papers describing new approaches to teaching programming or bemoaning the various problems. The problem is international [102]. There is nothing special about the universities of Leeds and Kent.

[3] The ideal strategy that would be used in industry is, of course, to show the code to some other programmers and to ask them to point out the error. In an academic context such an approach would be considered by some a dangerous first step on the road to plagiarism.

Even though the main focus has been on an examination of motivation, this study has shown once again some of the reasons why some students, even if apparently well prepared and well motivated at the start, find learning to program difficult. There are so many facets to this problem that it is difficult to know where to start but there is, perhaps, one common theme. In general the students do not *enjoy* the experience of learning to program. If they do not enjoy it they will not engage in the activities required of them. If they do not do that they will not reflect and they will not learn. It follows that a (perhaps *the*) key factor lurking behind the issues described in this section is the *motivation* of the students.

6.2 The Motivation of the Students

If nothing else has been achieved this study has certainly confirmed the suggestion that motivation is an extremely difficult concept to investigate. The students have answered questions about their motivation, and these responses have been interpreted, but it is not possible to be sure that the interpretation is correct. There remain the possibilities that students would be tempted to give the answers they believe to be 'correct' or even that some would through mischief give deliberately misleading answers. Nevertheless, if the interpretation is accepted a picture of the students' motivation has emerged.

The two essential components of motivation – value and expectancy – have been treated separately. The study of the entire class in chapter 4 throws light on the value students attach to their course while the more focused small-group study in chapter 5 looks more at their expectations. Both these components must be present for a student to be motivated and on the whole the news is good.

As regards value it is not a surprise to discover that the students do indeed value success in their studies. The surprise perhaps comes more from the reasons why they seem to value it. The initial survey showed that many (surprisingly many) were motivated by achievement, a fact that rather goes against the current stereotype of

the strategic students interested only in a future career. These students clearly exist (and future career was indeed the second most popular motivation) but they are far from the majority or norm. The later surveys showed something of a change in the reasons why the students valued success but there were no signs that they were ceasing to attach value in some form. The change was (again surprisingly) away from future career and toward achievement. This change was partly maintained in the final survey when achievement motivation was again the most popular.

So the students are motivated largely by the sense of achievement that they derive from doing well. While this is not related directly to their subject, it is essentially an intrinsic form of motivation, and a good thing. If they want to learn they will learn! It is clear that the value part of the motivation equation is at least positive (or, in slightly more mathematical terms, positive and non-zero). A slight concern is the increase over the study in the number of students claiming to just want to pass. This choice seems to reflect a lack of motivation or perhaps a problem in the expectancy element.

It could be argued that the different types of motivation used in this study are less than comparable. For example it might be suggested that achievement represents a short-term objective while motivations relating to a future career are clearly more long-term. These two may even represent different levels of motivation, with some forms of motivation corresponding to lower levels in Maslow's hierarchy. This said, at least the students are motivated in some way.

If the students are motivated it follows that a Biggs level 1 view of teaching (which holds that the students are basically unmotivated and lazy) is completely untenable. The problem in teaching programming has therefore moved at least to level 2. Lack of motivation (at least in terms of value) is not an issue.

Expectancy is more of a problem. There were clearly times in the module when some of the students did not expect to succeed. Max's "gonna get 0 – gonna fail – gonna die" is the most obvious case in point but there are many more. Overall there is a feeling that most of the students started the course with high hopes (or at the least

with a reasonable level of expectancy) even if some were a little nervous. During the course many (but happily not all) found their confidence gradually eroding until they no longer believed that they would pass. Assessment, and in particular feedback on assessment, has a crucial rôle here. There are clear signs that each assessment and the workload associated with it knocked back confidence and expectancy further and further. There are, of course, exceptions but enough of the study group showed this pattern for it to be taken as a significant issue.

If expectancy or value fall to zero the value of the other component does not matter and the student will not be motivated. Chapter 4 has shown that the value component is constant and positive (even if the reasons do seem to shift). However, chapter 5 has highlighted some clear problems with expectancy. A teacher must motivate a programming class. It follows from this that a teacher's primary focus should be the *expectancy* of the class. The teacher must be sure to satisfy the students' lowest level need. The students must believe that they will pass. This done, they can get on with the more important business of learning to program.

6.3 Motivating the Students

There are few books about learning to program.[4] One of the few is *Oh! Pascal!* by Doug Cooper and Mike Clancy. In the preface to the third edition [30] Cooper wrote "when I lecture I encourage any student who isn't so confident to make a smart friend, and to stick by her for the term.". This is an interesting quote. The student is lacking in confidence, not intelligence, and the lecturer's job is to encourage. This quote highlights the rôle that a teacher must adopt. The didactic instructor must become a sympathetic motivator.

If this approach is to work there must be a particular relationship between the teacher and the students. This relationship must be based on trust and, perhaps as part of

[4] There are many books about programming languages and some of these claim to be books about learning to program. They are not.

this, mutual respect. Unfortunately, many staff and students would prefer to operate in a 'them and us', theory X, climate. This might well be possible or even appropriate in some subjects or disciplines but surely it is not in programming. Programming is an unusual subject. It is best learned when the learners have ready access to a skilled programmer (presumably their teacher) for advice and support. This will not be the case in a theory X climate.

But how is a teacher to motivate a class? The first stage is obviously to acquire an understanding of the students' reasons for taking the course. Almost half of the students in this study saw the programming class as simply a compulsory part of their degree. It was little more than an academic hoop through which they were required to jump. This does not point to motivation to learn or to any form of interest. The teacher must explain why the students should be interested and must gain their enthusiasm. Above all, this must happen before anyone goes anywhere near any programming syntax.

Assessment has a powerful effect on motivation. Some students in a class will be motivated particularly to do well in assessments while other students will instead find their motivation deflated by failure. There are plentiful examples of both in chapter 5. There is often too much assessment in programming courses, often perhaps deriving from a level 1 "if I don't assess it they won't do it" view. This is terribly wrong. The students in this study are almost not following a programming course. They are lurching unhappily from one programming assessment to the next. If the teacher wants them to reflect on an assessment (and therein lies the true value of the exercise) they can hardly be expected to do so if there is too much assessment. The students have other subjects to study. There must be summative assessment, of course, but there should be the minimum necessary to ensure that standards are met. Far more valuable are well designed formative assessments. This is a type of assessment that can have a powerful and beneficial motivating effect.

The idea that programming can be taught in lectures is a quaint one. Experienced programmers approaching a new language do not immediately seek out a lecture

course. They acquire a book, try a few simple programs, perhaps chat to an expert, and gradually the book becomes a reference. Why can student programmers not learn in the same way? There is no reason at all. The 'lecturer' could easily take on the job of choosing a book, prescribing some weekly readings, providing some suitable examples, and would then take on the expert rôle. This is a tried-and-tested scheme in the commercial world so why should it not be used or at least tried in universities?

In such a model the teacher's rôle is very much that of a motivator. Yes. Learning to program is difficult but it is not impossible. A teacher's job is to reassure students (to address expectancy) and to provide occasional help. This rôle involves no-one in entering a lecture theatre.

If teachers are to teach students to program effectively they must become motivators. Motivation is not taking place in lecture theatres.

6.4 Future Work

This study has raised some interesting questions and many of these could and should form the basis of future work.

The data collected, especially that for the two classes at Leeds and Kent, could be analysed in different ways to address questions such as:

- Are the trends in motivation significantly different at the two institutions?
- Are there any differences between the attitudes of male and female students?
- Or between home and overseas students?
- Or between traditional 18-year-old entrants and mature students?
- How do the motivations of students with a particular initial motivation change over the course? Are there any trends visible?

Appendix D presents a very brief comparison between the results at Leeds and Kent (which might form a start in considering the first question) and more of the raw data is presented in appendix E.

The students followed in chapter 5 have now just completed the second year of their courses. It would be interesting to revisit their experience of programming with them. Do the feelings that they revealed about programming during this study persist? Do they now seem to think in the same way as the veterans did here? All these students were studying the same course at the same institution. It would certainly be interesting to follow other groups of students to investigate whether or not the experiences at other institutions were comparable.

This work has touched once again on the thorny issue of whether or not mathematical skills or attainment have any influence on programming ability or on the ability to learn to program. It is possible to find studies that hold that mathematical ability is a good measure of aptitude for programming and equally possible to find studies that present quite the opposite opinion. This is certainly worthy of further investigation, not least to inform admissions procedures and requirements.

The two institutions considered here are essentially traditional UK universities. A wider study at more institutions in the UK, ideally including some of the 'post-1992' universities,[5] would enable the findings to be extended or refined. A comparison with part-time or distance students (where the motivational issues are rather different [71]) would also broaden the findings. A more ambitious study would be to extend the work beyond the UK to see if the same issues are present in other countries. There is already evidence that the problems are shared [102].

There has been no attempt in this study to separate the factors that are generic to all students from those that are more specific to programming students. It would be interesting and informative to try to identify these and separate them out. The *UNIQoLL*[6] [150] project (started in the School of Computing at the University of Leeds) considers student well-being using a variety of metrics [37]. This project has

[5] These are universities that were, until 1992, polytechnics or similar colleges. They tended to offer more vocational courses and usually had lower entry standards than universities.

[6] University National Initiative on Quality of Life and Learning.

now been extended to the entire university and it is now possible to separate trends in student well-being according to degree programme. It would be interesting to combine the results from this study with the UNIQoLL data for the same cohort of students. Such work might seek to provide answers to questions such as:

- What are the additional pressures faced by programming students?
- Are programming students in any sense worse off than students not studying programming?
- Do the observable changes in student well-being match any observed changes in motivation?

The teaching of programming at both Leeds and Kent continues to evolve. It would be very interesting to repeat this study after the teaching has been refined in some way. For one thing this would serve as a way to evaluate the effectiveness of the change. This work has been in many ways only a start.

6.5 Reflections

The end of any substantial piece of work such as this presents a fine opportunity for some reflection. There follow some thoughts on the process of carrying out this work and what, perhaps, could have been done better. It will not be a surprise that these are not presented in any particular order of importance.

The questionnaires delivered to the complete classes were designed for ease of analysis. They were indeed easy to analyse but this was not without its costs. The method was at times too crude and it was sometimes evident that some students had a more complicated tale to tell than the questionnaire was allowing them to record. Some of them indicated this and there were no doubt others who did not. A more free-form approach, perhaps by providing some factors to rank rather than to choose from, would have provided a richer set of primary data.

The questionnaires were anonymous. This meant that it was not possible to follow up any of the particularly interesting responses. It was also not possible to compare the responses to the students' final grades (something that might have formed the basis of an interesting comparison between motivation, attitudes, and final result). Less anonymous questionnaires would have allowed each student's changing views to be tracked through the course but it is possible that less anonymous questionnaires would have produced different responses if the students felt in some sense more accountable for their answers.

The distribution of the questionnaires in lectures may also have skewed the findings. The detailed results (appendix E) clearly show how the number of responses decreased with each questionnaire (especially at Kent). However, it is hard to think of an equally convenient method that would have provided better numbers of returns.

The process of following the novice students through their course was an interesting one. It is fascinating that several of them were recording on their weekly sheets experiences and emotions that were quite at odds with those they were presenting in public. If only each could have seen what the others were writing! This is perhaps understandable since during the first semester at university most students are trying to find their feet and are not sure where they fit in. Looking at their private thoughts during the semester would to an extent have defeated the object of the study by introducing an artificial intervention but it certainly might have meant that they would have got the help they needed sooner.

This work has involved the study of a great deal of literature. There is an immense amount of educational literature and so much of it seems to be so relevant. It is disappointing that so much of it seems to be rooted firmly in theory and is so far away from the practicalities of UK higher education today. There was at times depressingly little that was directly relevant to the issues at hand. There is also, of course, an increasing literature in computing (some call it computer science) education. So much of this is also disappointing. It rarely draws on the educational theory and many papers amount to little more than descriptions of courses or novel methods of assessment. There are honourable exceptions but they are exceptions.

Motivation itself has proved to be a difficult business. It is an inherently abstract concept, impossible to measure and difficult to identify. This is unlike so much else in the field of computing where an algorithm can be proved categorically to be correct, a database can be shown to be correctly normalised, or a problem can be shown to be insoluble. Motivation is not like this. It is possible only to enquire, observe, and make suggestions or assumptions. Some of the work presented here may seem unscientific for this reason but that is inevitable when dealing with so abstract a concept. It is also possible that at times the findings are the result of over-enthusiasm and are taken beyond what is shown by the data. If any instances of this remain, apologies are offered. The raw data are included in appendices and wait there for further interpretation.

This has indeed been an interesting exercise.

6.6 The Final Word

I[7] talk to many students during their first year. If I mention their academic progress the talk will always turn to programming. More often than not the programming course is seen as a nightmare that has (hopefully) passed and the students are keen to see how they can avoid programming in the future. I have on several occasions seen students leave the university simply to avoid programming. Every year I see students spending untold hours on their assignments. I see misery and suffering.

I confess that I had formed the view that these students were the norm. During this work I have met other students. My teaching rôle in the programming course has always been 'Master of the Novices'. I look after the strugglers and persevere with the hopeless cases. This work has given me the chance to meet other students. I have met some who take to programming quite easily and enjoy it. I had heard rumours about these people but I am not convinced that I believed in them. It was something of a relief to discover that they did indeed exist.

[7] The author is afraid that he is going to lapse into the first person again.

This said, though, I have also found many more students who matched only too well my previous experience. There are more than a few of them (the number is impossible to determine) and they genuinely suffer. This cannot be right. One topic should not dominate the curriculum and the student experience to this extent. It has to be the case that there is something fundamentally wrong with the way in which we teach programming. One step (and I do not claim a panacea) is to appreciate the crucial rôle that motivation has to play in teaching programming. Students will not learn unless they are motivated. It must be a teacher's main task, therefore, to ensure that all the students are properly motivated.

Undertaking this study has been an interesting experience. For the first time in many years I have been able to take an outsider's view of the programming course (even if I could not resist joining in with the occasional lab session). My views have been challenged and have been changed. I started this work two years ago with the idea that the students were not motivated to learn programming (yes, level 1!) and that what was needed was some new instructional techniques. I planned to throw Frisbees in even more interesting ways. I think that I now realise just how wrong I was. I have learned much about the ways in which students learn to program and I have thought much about the ways in which they are taught. I can only hope that in some way I am now a better teacher.

I am, the students tell me, no longer 'Master of the Novices'. I am now their 'Morale Officer'. We will see what happens.

Thanks for reading. This was fun. Like history [140], this work has now also come to a .

References

[1] Archer, J. Encouraging students' motivation to learn. On-line at http://www.newcastle.edu.au/oldsite/services/iesd/publications/eunexus/articles/teaching_guides/encouraging/encouraging_1.htm.

[2] Astrachan, O. Hooks and props in teaching programming. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 21–24.

[3] Baldwin, D. Discovery learning in computer science. In *Proceedings of SIGCSE '96, Philadelphia, USA* (1996), ACM, pp. 222–226.

[4] Ball, S., Ed. *Motivation in Education.* Academic Press, 1977.

[5] Barnes, D. J. Public forum help seeking: The impact of providing anonymity on student help seeking behaviour. In *Computer Based Learning in Science '99*, G. M. Chapman, Ed. CBLIS, 1999.

[6] BBC News. BBC News 22nd April 2002. Student debt 'tops £10,000'. On-line at http://news.bbc.co.uk/hi/english/education/newsid_1939000/1939528.stm.

[7] BBC News. BBC News 22nd May 2002. University a goal for many pupils. On-line at http://news.bbc.co.uk/hi/english/education/newsid_2002000/2002531.stm.

[8] BBC News. BBC News 22nd October 2001. Universities ordered to widen access. On-line at http://news.bbc.co.uk/hi/english/education/newsid_1612000/1612614.stm.

[9] BBC News. BBC News 25th July 2001. Students face growing debt burden. On-line at `http://news.bbc.co.uk/hi/english/education/newsid_1456000/1456269.stm`.

[10] BBC News. BBC News 4th March 2002. Getting students to work. On-line at `http://news.bbc.co.uk/hi/english/education/newsid_1849000/1849549.stm`.

[11] Ben-Ari, M. Constructivism in computer science education. In *Proceedings of SIGCSE '98, Atlanta, USA* (1998), ACM, pp. 257–261.

[12] Bereiter, C., and Ng, E. Three levels of goal orientation in learning. *Journal of the Learning Sciences 1* (1991), 243–271.

[13] Bereiter, C., and Scardamalia, M. *Surpassing Ourselves – An Inquiry into the Nature and Implications of Expertise*. Open Court, 1993.

[14] Biggs, J. *Teaching for Quality Learning at University*. Society for Research into Higher Education, 1999.

[15] Bloom, B. S. *Taxonomy of Educational Objectives: The Classification of Educational Goals*. Longman, 1956.

[16] Boyle, R. D., Carter, J., and Clark, M. What makes them succeed? Entry, progression and graduation in computer science. *Journal of Further and Higher Education 26* (2002), 3–18.

[17] Boyle, R. D., Jackson, J., and Wade, R. Changing learning culture with electronic bulletin boards. Research Report 95.02, School of Computing, University of Leeds, 1995.

[18] Brilliant, S. S., and Wiseman, T. R. The first programming paradigm and language dilemma. In *Proceedings of SIGCSE '96, Philadelphia, USA* (1996), ACM, pp. 338–342.

[19] Brown, G., and Atkins, M. *Effective Teaching in Higher Education*. Routledge, 1988.

[20] Brown, S. Institutional strategies for assessment. In *Assessment Matters in Higher Education*, S. Brown and A. Glasner, Eds. Society for Research into Higher Education, 1999, pp. 3–13.

[21] Byrne, P., and Lyons, G. The effect of student attributes on success in programming. In *Proceedings of ITiCSE 2001, Canterbury, UK* (2001), ACM, pp. 49–52.

[22] Carter, J., and Jenkins, T. Gender and programming: What's going on? In *Proceedings of ITiCSE '99, Krakow, Poland* (1999), ACM, pp. 1–4.

[23] Carter, J., and Jenkins, T. Gender differences in learning to program. *Computing Research News 12* (2000), 2 and 14.

[24] Carter, J., and Jenkins, T. Redressing the decline – How can we encourage women back into computer science? In *Proceedings of Higher Education Close Up 2, University of Lancaster* (2001), pp. 6–8.

[25] Carter, J., and Jenkins, T. Where have all the girls gone? What entices female students to apply for a computer science degree. In *Proceedings of 2nd Annual LTSN for Information and Computer Science Conference, London, UK* (2001), Learning and Teaching Support Network for Information and Computer Science, pp. 72–77.

[26] Carter, J., Tardivel, J., Fincher, S., Fuller, U., Johnson, C., Linington, J., and Utting, I. Portrait of 2000/01 part I assessments, part 1: Statistical analysis. Technical Report 10-01, Computing Laboratory, University of Kent at Canterbury.

[27] Chamillard, A. T., and Karolick, D. Using learning style data in an introductory computer science course. *SIGCSE Bulletin 31* (1999), 291–295.

[28] Clark, M., and Jenkins, T. What are they going to do now? The expectations and intentions of new information systems undergraduates. In *UKAIS '99: Information Systems The Next Generation*, L. Brooks and C. Kimble, Eds. McGraw-Hill, 1999, pp. 755–764.

[29] Colsant, Jr., L. C. "Hey, man, why do we gotta take this...?" Learning to listen to students. In *Reasons for Learning: Expanding the Conversation on Student-Teacher Collaboration*, J. G. Nicholls and T. A. Thorkildsen, Eds. Teachers College Press, 1995, pp. 62–89.

[30] Cooper, D. *Oh! Pascal!*, third ed. Norton, 1993.

[31] Cowan, J. *On Becoming an Innovative University Teacher*. Society for Research into Higher Education, 1998.

[32] Cox, B. *Practical Pointers for University Teachers*. Kogan Page, 1994.

[33] Culwin, F. Objects first, objects last or objects at all? In *Proceedings of the Fifth All Ireland Conference on the Teaching of Computing* (1997), Centre for Teaching Computing, Dublin City University.

[34] Curzon, P., and Rix, J. Why do students take programming modules? In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 59–63.

[35] Dall'Alba, G. Foreshadowing conceptions of teaching. *Research and Development in Higher Education 13* (1991), 293–297.

[36] Davis, H. C., Carr, L., Cooke, E., and White, S. Managing diversity: Experiences teaching programming principles. In *Proceedings of 2nd Annual LTSN for Information and Computer Science Conference, London, UK* (2001), Learning and Teaching Support Network for Information and Computer Science, pp. 53–59.

[37] Davy, J. R., Audin, K., Barkham, M., and Joyner, C. Student well-being in a computing department. In *Proceedings of ITiCSE 2000, Helsinki, Finland* (2000), ACM, pp. 136–139.

[38] Davy, J. R., and Jenkins, T. Research-led innovation in teaching and learning programming. In *Proceedings of ITiCSE '99, Krakow, Poland* (1999), ACM, pp. 5–8.

[39] DeCharms, R. *Personal Causation: The Internal Effective Determinants of Behavior*. Academic Press, 1968.

[40] Deci, E. E., and Ryan, R. M. *Intrinsic Motivation and Self-Determination in Human Behavior*. Plenum Press, 1985.

[41] Deitel, H. M., and Deitel, P. J. *C++ How to Program*, second ed. Prentice-Hall, 1998.

[42] Deitel, H. M., and Deitel, P. J. *C++ How to Program*, third ed. Prentice-Hall, 2001.

[43] Dewey, J. *Experience and Education*. MacMillan, 1938.

[44] Dijkstra, E. W. On the cruelty of really teaching computer science. *Communications of the ACM 32* (1989), 1398–1404.

[45] du Boulay, B. Some difficulties of learning to program. In *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Lawrence Erlbaum Associates, 1989, pp. 283–299.

[46] du Boulay, B., O'Shea, T., and Monk, J. The black box inside the glass box: Presenting computing concepts to novices. In *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Lawrence Erlbaum Associates, 1989, pp. 431–446.

[47] Dunn, B. An exploration of student experiences and ethos through narratives. In *Proceedings of Qualitative Evidence-Based Practice Conference* (2000), Coventry University.

[48] Earwaker, J. *Helping and Supporting Students*. Society for Research into Higher Education, 1992.

[49] Elton, L. Strategies to enhance student motivation: A conceptual analysis. *Studies in Higher Education 21* (1996), 57–68.

[50] Entwistle, N. Motivation and approaches to learning: Motivating and conceptions of teaching. In *Motivating Students*, S. Brown, S. Armstrong, and G. Thompson, Eds. Kogan Page, 1998, pp. 15–23.

[51] ETH Zürich. ETH Oberon home page. On-line at `http://www.oberon.ethz.ch/`.

[52] Evans, G. E., and Simkin, M. G. What best predicts computer proficiency? *Communications of the ACM 32* (1989), 1322–1327.

[53] Fallows, S., and Ahmet, K. *Inspiring Students: Case Studies in Motivating the Learner*. Kogan Page, 1999.

[54] Fell, H. J., and Proulx, V. K. Exploring Martian planetary images: C++ exercises for CS1. In *Proceedings of SIGCSE '97, San Jose, USA* (1997), ACM, pp. 30–34.

[55] Fincher, S. What are we doing when we teach programming? In *Frontiers in Education '99* (1999), IEEE, pp. 12a41–5.

[56] Fleury, A. Acting out algorithms: How and why it works. In *Proceedings of the 4th Annual CCSC Midwestern Conference* (1997). Also published in *Journal of Computing in Small Colleges*, 13:83–90, 1997.

[57] Goold, A., and Rimmer, R. Factors affecting performance in first-year computing. *SIGCSE Bulletin 32* (2000), 39–43.

[58] Gray, J., Boyle, T., and Smith, C. A constructivist learning environment implemented in Java. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 94–97.

[59] Grayson, A., Clarke, D. D., and Miller, H. Help-seeking among students: Are lecturers seen as a potential source of help? *Studies in Higher Education 23* (1998), 143–155.

[60] Greasley, K. Does gender affect students' approaches to learning? In *Motivating Students*, S. Brown, S. Armstrong, and G. Thompson, Eds. Kogan Page, 1998, pp. 105–112.

[61] Greening, T. Students seen flocking in programming assignments. In *Proceedings of ITiCSE 2000, Helsinki, Finland* (2000), ACM, pp. 93–96.

[62] Guckes, S. The VIM home page. On-line at `http://www.vim.org/`.

[63] Habeshaw, S., Gibbs, G., and Habeshaw, T. *53 Problems with Large Classes*. TES Associates, 1992.

[64] Hagan, D., and Markham, S. Does it help to have some programming experience before beginning a computing degree program? In *Proceedings of ITiCSE 2000, Helsinki, Finland* (2000), ACM, pp. 25–28.

[65] Hagan, D., and Sheard, J. The value of discussion classes for teaching introductory programming. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 108–111.

[66] Herzberg, F., Mausner, B., and Snyderman, B. B. *The Motivation to Work*. Wiley, 1959.

[67] Higginbotham, T. F. Converting experienced C++ programmers to Java. Presented at the Sixth Java and the Internet in the Computing Curriculum Conference, University of North London, January 2002. Available on-line at `http://www.ics.ltsn.ac.uk/pub/jicc6/`.

[68] Holmes, G., and Smith, T. C. Adding some spice to CS1 curricula. In *Proceedings of SIGCSE '97, San Jose, USA* (1997), ACM, pp. 204–208.

[69] Howard, R. A., Carver, C. A., and Lane, W. D. Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: Tying it all together in the CS2 course. In *Proceedings of SIGCSE '96, Philadelphia, USA* (1996), ACM, pp. 227–231.

[70] Hughes, S. Strategies adopted by academic staff in a mass higher education system. In *Proceedings of Higher Education Close Up, University of Central Lancashire, July 1998* (1998).

[71] Isroff, K., and del Soldato, T. Students' motivation in higher education contexts. In *Motivating Students*, S. Brown, S. Armstrong, and G. Thompson, Eds. Kogan Page, 1998, pp. 73–82.

[72] Jacobs, P. A., and Newstead, S. E. The nature and development of student motivation. *British Journal of Educational Psychology 70* (2000), 243–254.

[73] Jenkins, T. A participative approach to teaching programming. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 125–129.

[74] Jenkins, T. The motivation of students of programming. In *Proceedings of ITiCSE 2001, Canterbury, UK* (2001), ACM, pp. 53–56.

[75] Jenkins, T. Teaching programming – A journey from teacher to motivator. In *Proceedings of 2nd Annual LTSN for Information and Computer Science Conference, London, UK* (2001), Learning and Teaching Support Network for Information and Computer Science, pp. 65–71.

[76] Jenkins, T. *Learning to Program (with C++).* Palgrave Macmillan, 2002. In press.

[77] Jenkins, T., and Davy, J. R. Dealing with diversity in introductory programming. In *Proceedings of 1st Annual LTSN for Information and Computer Science Conference, Edinburgh, Scotland* (2000), Learning and Teaching Support Network for Information and Computer Science, pp. 81–87.

[78] Jenkins, T., and Davy, J. R. Diversity and motivation in introductory programming. *ITALICS (Innovations in Teaching and Learning in Information and Computer Sciences) 1* (2001). On-line at `http://www.ics.ltsn.ac.uk/pub/italics/issue1/tjenkins/003.html`.

[79] Jenkins, T., and Gillespie, C. *Learning to Program (with Java)*. Palgrave Macmillan, 2003. In press.

[80] Jenkins, T., and Towle, W. Teaching programming to novices – Can technology help? In *Proceedings of the Fifth All Ireland Conference on the Teaching of Computing* (1997), Centre for Teaching Computing, Dublin City University.

[81] Keller, J. M. Motivational design of instruction. In *Instructional-Design Theories and Models: An Overview of their Current Status*, C. M. Reigeluth, Ed. Lawrence Erlbaum Associates, 1983, pp. 383–434.

[82] Kernighan, B. W., and Ritchie, D. M. *The C Programming Language*. Prentice-Hall, 1978.

[83] King, C. Pay as you learn? Students in the changing university. In *The Changing University?*, T. Schuller, Ed. Society for Research into Higher Education, 1995, pp. 116–127.

[84] Kneale, P. E. The rise of the "strategic student": How can we adapt to cope? In *Facing up to Radical Changes in Universities and Colleges*, S. Armstrong, G. Thompson, and S. Brown, Eds. Kogan Page, 1997, pp. 119–130.

[85] Knight, P. T., and Trowler, P. R. Department-level cultures and the improvement of learning and teaching. *Studies in Higher Education 25* (2000), 69–83.

[86] Kolb, D. *Experiential Learning: Experience as the source of learning and development*. Prentice-Hall, 1985.

[87] Kolb, D. *Learning Style Inventory.* McBer, 1985.

[88] Kölling, M. BlueJ – Teaching Java. On-line at `http://www.bluej.org/`.

[89] Laurillard, D. *Rethinking University Teaching.* Routledge, 1993.

[90] Lefcourt, H. M. *Locus of Control: Current Trends in Theory and Research.* Lawrence Erlbaum Associates, 1976.

[91] Lewandowski, G., and Morehead, A. Computer science through the eyes of dead monkeys: Learning styles and interaction in CS1. In *Proceedings of SIGCSE '98, Atlanta, USA* (1998), ACM, pp. 312–316.

[92] Lewis, S., and Mulley, G. A comparison between novices and experienced compiler users in a learning environment. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 157–161.

[93] Liffick, B. W., and Aiken, R. A novice programmer's support environment. In *Proceedings of ITiCSE '96, Barcelona, Spain* (1996), ACM, pp. 49–51.

[94] Linn, M. C., and Clancy, M. J. The case for case studies of programming problems. *Communications of the ACM 35* (1992), 121–132.

[95] Lister, R. Objectives and objective assessment in CS1. In *Proceedings of SIGCSE 2001, Charlotte, USA* (2001), ACM, pp. 292–296.

[96] MacFarlane, A. G. J., Ed. *Teaching and Learning in an Expanding HE System (the 'MacFarlane Report').* Committee of Scottish University Principals, 1992.

[97] Machiavelli, N. *The Prince.* Philip Allan, 1925.

[98] Marton, F., and Säljö, R. On qualitative differences in learning I: Outcome and process. *British Journal of Educational Psychology 46* (1976), 4–11.

[99] Maslow, A. H. *Motivation and Personality.* Harper, 1954.

[100] Mayer, R. E. The psychology of how novices learn computer programming. In *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Lawrence Erlbaum Associates, 1989, pp. 129–159.

[101] Mazlack, L. J. Identifying potential to acquire programming skill. *Communications of the ACM 23* (1980), 14–17.

[102] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. A multi-national, multi-institutional study of assessment of programming skills of first year CS students. *SIGCSE Bulletin 33* (2001), 125–140.

[103] McGregor, D. *The Human Side of Enterprise*. McGraw-Hill, 1960.

[104] McKeachie, W. J. Learning styles can become learning strategies. *The National Teaching & Learning Forum 4* (1995), 1–3.

[105] McKeithen, K., Reitman, J. S., Reuter, H. H., and Hirtle, S. C. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology 13* (1981), 307–325.

[106] Mercer, R. *Computing Fundamentals with C++*. Macmillan, 1999.

[107] Mercuri, R., Hermann, N., and Popyack, J. Using HTML and JavaScript in introductory programming courses. In *Proceedings of SIGCSE '98, Atlanta, USA* (1998), ACM, pp. 176–179.

[108] Merriam-Webster Inc. Merriam-Webster OnLine. On-line at `http://www.m-w.com/`.

[109] Metrowerks Inc. Metrowerks, Inc. On-line at `http://www.metrowerks.com/`.

[110] Moodie, G. *Standards and Criteria for Higher Education*. Society for Research into Higher Education, 1986.

[111] Moore, J. Singing to students – A study in entertaining education. Final Year Project, School of Computing, University of Leeds, 2001.

[112] Moser, R. A fantasy adventure game as a learning environment: Why learning to program is so difficult and what can be done about it. In *Proceedings of ITiCSE '97, Uppsala, Sweden* (1997), ACM, pp. 114–116.

[113] Mowl, G. Innovative student assessment. On-line at `http://www.lgu.ac.uk/deliberations/assessment/mowl.html`.

[114] Newstead, S. Individual difference in student motivation. In *Motivating Students*, S. Brown, S. Armstrong, and G. Thompson, Eds. Kogan Page, 1998, pp. 189–199.

[115] Nosek, J. T. The case for collaborative programming. *Communications of the ACM 41* (1998), 105–108.

[116] November, P. Learning to teach experientially: A pilgrim's progress. *Studies in Higher Education 22* (1997), 289–299.

[117] Nulty, D. D., and Barrett, M. A. Transitions in students' learning styles. *Studies in Higher Education 21* (1996), 333–345.

[118] Orne, M. T. On the social psychology of the psychological experiment: With particular reference to demand characteristics and their implications. *American Psychologist 17* (1962), 776–783.

[119] Oxford English Dictionary. Oxford English Dictionary On-line. On-line at `http://www.oed.com/`.

[120] Palfrey, C., and Roberts, A. *The Unofficial Guide to Wales*. Y Lolfa, 1994.

[121] Parnas, D. L. Response to Dijkstra's Article [44]. *Communications of the ACM 32* (1989), 1405–1406.

[122] Pask, G. Styles and strategies of learning. *British Journal of Educational Psychology 46* (1976), 12–15.

[123] Perkins, D. N., Schwartz, S., and Simmons, R. Instructional strategies for the problems of novice programmers. In *Teaching and Learning Computer Programming*, R. E. Mayer, Ed. Lawrence Erlbaum Associates, 1988, pp. 153–178.

[124] Peterson, C., Maier, S. F., and Seligman, M. E. P. *Learned Helplessness: A Theory for the Age of Personal Control*. Oxford University Press, 1993.

[125] Pratchett, T. *Interesting Times*. Victor Gollancz, 1994.

[126] Prosser, M., and Trigwell, K. *Understanding Learning and Teaching*. Society for Research into Higher Education, 1999.

[127] Protz, R., Ed. *The Good Beer Guide 2002*. Campaign for Real Ale, 2001.

[128] Ramsden, P., and Entwistle, N. J. Effects of academic departments on students' approaches to studying. *British Journal of Educational Psychology 51* (1981), 368–383.

[129] Rankin, A., Ed. *The Leeds Beer Drinkers' Companion*. The Leeds Branch of the Campaign for Real Ale, 2002.

[130] Red Mole. The Red Mole Alternative Guide to Universities. On-line at `http://www.redmole.co.uk/`.

[131] Reek, M. M. A top-down approach to teaching programming. In *Proceedings of SIGCSE '95, Nashville, USA* (1995), ACM, pp. 6–9.

[132] Rickinson, B. The relationship between undergraduate counselling and successful degree completion. *Studies in Higher Education 23* (1998), 95–102.

[133] Roberts, E., Lilly, J., and Rollins, B. Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience. In *Proceedings of SIGCSE '95, Nashville, USA* (1995), ACM, pp. 48–52.

[134] Rowland, S. *The Enquiring University Teacher.* Society for Research into Higher Education, 2000.

[135] Rowntree, D. *Assessing Students: How shall we know them?* Kogan Page, 1987.

[136] Schank, R. C. Revolutionizing the traditional classroom course. *Communications of the ACM 44* (2001), 21–24.

[137] Schmeck, R. R. Strategies and styles of learning. In *Learning Strategies and Learning Styles*, R. R. Schmeck, Ed. Plenum Press, 1988, pp. 317–347.

[138] Schoenfeld, A. H. *Cognitive Science and Mathematics Education.* Lawrence Erlbaum Associates, 1987.

[139] Schön, D. A. *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions.* Jossey-Bass, 1987.

[140] Sellars, W. C., and Yeatman, R. J. *1066 and All That.* Methuen, 1931.

[141] Seltzer, S. The IJDb Video Database. On-line at `http://www.jugglingdb.com/videos/`.

[142] Shea, R., and Wilson, R. A. *The Illuminatus! Trilogy.* Raven, 1998.

[143] Sheard, J., and Hagan, D. Our failing students: A study of a repeat group. In *Proceedings of ITiCSE '98, Dublin, Ireland* (1998), ACM, pp. 223–227.

[144] Siegel, E. V. Why do fools fall into infinite loops: Singing to your computer science class. In *Proceedings of ITiCSE '99, Krakow, Poland* (1999), ACM, pp. 167–170.

[145] Sloane, K. D., and Linn, M. C. Instructional conditions in Pascal programming classes. In *Teaching and Learning Computer Programming*, R. E. Mayer, Ed. Lawrence Erlbaum Associates, 1988, pp. 207–235.

[146] Spohrer, J. C., and Soloway, E. Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM 29* (1986), 624–632.

[147] Spohrer, J. C., and Soloway, E. Novice mistakes: Are the folk wisdoms correct? In *Studying the Novice Programmer*, E. Soloway and J. C. Spohrer, Eds. Lawrence Erlbaum Associates, 1989, pp. 401–416.

[148] Thomas, D. *Under Milk Wood*. Dent, 1954.

[149] Tynjälä, P. Traditional studying for examination versus constructivist learning tasks: Do the outcomes differ? *Studies in Higher Education 23* (1998), 173–189.

[150] UNIQoLL Project. UNIQoLL. On-line at `http://www.comp.leeds.ac.uk/uniqoll/`.

[151] van Rossum, G., et al. Computer Programming for Everybody. On-line at `http://www.python.org/cp4e/`.

[152] Webb, N. M., and Lewis, S. The social context of learning computer programming. In *Teaching and Learning Computer Programming*, R. E. Mayer, Ed. Lawrence Erlbaum Associates, 1988, pp. 179–206.

[153] Weber-Wulff, D. Combating the code warrior: A different sort of programming instruction. In *Proceedings of ITiCSE 2000, Helsinki, Finland* (2000), ACM, pp. 85–88.

[154] Weinberg, G. M. *The Psychology of Computer Programming*. Van Nostrand Reinhold, 1971.

[155] Wilcox, S. Fostering self-directed learning in the university setting. *Studies in Higher Education 21* (1996), 165–176.

[156] Willcoxson, L. The impact of academics' learning and teaching preferences on their teaching practices: A pilot study. *Studies in Higher Education 23* (1998), 59–70.

[157] Williams, L. A. Pair programming experiment. On-line at `http://www.extremeprogramming.org/stories/pair6.html`.

[158] Williams, L. A., and Kessler, R. R. All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM 43* (2000), 108–114.

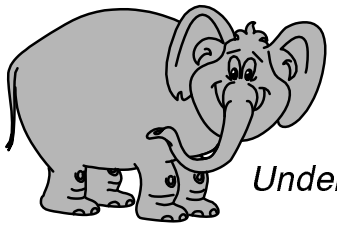
[159] Wirth, N. The programming language Pascal. *Acta Informatica 1* (1971), 35–63.

Appendix A

Class Questionnaires

The following pages contain the questionnaires distributed to the classes at Leeds and Kent as they progressed through their course. They were distributed as they are presented here and with no introduction or instructions. The responses to these questionnaires formed the basis of the discussions in chapter 4.

The elephant is something of a mystery, its origins clouded in the mists of time.



ProgyQuoL

Understanding the First Programming Experience

This project aims to understand the experience of learning to program for the first time. Programming is an unusual academic subject in that different people learn it best in very different ways. By better understanding the ways in which different types of people learn, we hope to be able to provide a better learning experience for all.

Thanks for your time. Any queries on this work can be sent to tony@comp.leeds.ac.uk.

Please provide the following information about yourself by circling the answer that applies:

Age: **Under 21** **Over 21**
Gender: **Female** **Male**
Origin: **UK** **EU** **Overseas**

Where are you studying? **Kent** **Leeds**

What degree programme are you studying?

Is the programming part of your course compulsory or optional for you?
Compulsory **Optional**

*Please write here the **one word** that best describes your reason for taking your degree programme:*

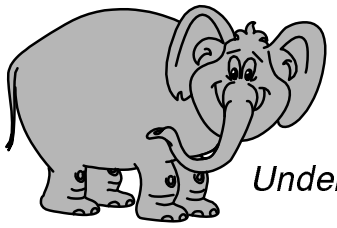
*Please write here the **one word** that best describes your reason for taking this programming module:*

Which one of these statements best describes your attitude to your degree programme? Please tick one.

❒ I want to do well for my own satisfaction.
❒ I want to do well to please my parents or family.
❒ I want to do well to please my teachers.
❒ I want to do well so that I will be able to get a good job.
❒ I just want to pass.

Form 1 - Class, Before

Figure A1: Questionnaire – Before the Module



ProgyQuoL

Understanding the First Programming Experience

A long time ago you probably complete a questionnaire with an elephant on it. Would you mind too much filling in just one more? Many thanks.

This project aims to understand the experience of learning to program for the first time. Programming is an unusual academic subject in that different people learn it best in very different ways. By better understanding the ways in which different types of people learn, we hope to be able to provide a better learning experience for all.

Please provide the following information about yourself by circling the answer that applies:

Age: **Under 21** **21 or Over**
Gender: **Female** **Male**
Origin: **UK** **EU** **Overseas**

Where are you studying? **Kent** **Leeds**

What degree programme are you studying?

Is the programming part of your course compulsory or optional for you?
Compulsory **Optional**

*Please write here the **one word** that best describes your attitude today towards the C++ or Java programming course you have done this semester:*

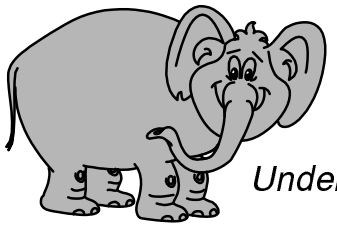
*Please write here the **one word** that best describes your attitude today towards the programming course you will take next semester:*

Which one of these statements best describes your attitude to your degree programme as a whole? Please tick just one.

❒ I want to do well for my own satisfaction.
❒ I want to do well to please my parents or family.
❒ I want to do well to please my teachers.
❒ I want to do well so that I will be able to get a good job.
❒ I just want to pass.

Form 7 - Class, After

Figure A2: Questionnaire – The Halfway Point



ProgyQuoL

Understanding the First Programming Experience

This project aims to understand the experience of learning to program for the first time. Programming is an unusual academic subject in that different people learn it best in very different ways. By better understanding the ways in which different types of people learn, we hope to be able to provide a better learning experience for all.

Thanks for your time. Any queries on this work can be sent to tony@comp.leeds.ac.uk.

Please provide the following information about yourself by circling the answer that applies:

Age: **Under 21** **21 or Over**
Gender: **Female** **Male**
Origin: **UK** **EU** **Overseas**

Where are you studying? **Kent** **Leeds**

What degree programme are you studying?

*Please write here the **one word** that best describes your attitude today towards the C++ or Java programming course you have done this year.*

Which one of the following statements best describes your attitude to programming now?

❒ Programming is fine. I can do it.
❒ Programming is OK. I can get by, but I don't enjoy it.
❒ I never want to do programming again.

Can you see yourself working as a programmer in the future?

Yes ❒ No ❒

Which one of these statements best describes your attitude to your degree programme as a whole? Please tick just one.

❒ I want to do well for my own satisfaction.
❒ I want to do well to please my parents or family.
❒ I want to do well to please my teachers.
❒ I want to do well so that I will be able to get a good job.
❒ I just want to pass.

Form 8 - Class, End

Figure A3: Questionnaire – After the Module

Appendix B

Words and Categories

The key to the analysis of the one-word answer questions used in the surveys of classes at Leeds and Kent (described in chapter 4) is the category to which each word was assigned. The questions were completely free-form and so there was a wide variety of responses. This appendix lists all the words given in responses and the category to which each was assigned.

The words in each section are simply in alphabetical order.

Motivation for Degree Programme

The following sections list all the words corresponding to the categories of response to the first question ('Why are you taking this degree programme?') in the survey of the class before they had started their course. This is a complete version of the description in section 4.2.2 on page 93.

Achievement Words – *Achievement, Ambition, Ambitious, Challenge, Challenging, Experience, Myself, Qualification, Satisfaction, Self-Actualisation, Success.*

Aspiration Words – *Advancement, Avarice, Career, Employability, Employment, Future, Greed, Job, Money, Necessary, Opportunities, Opportunity, Prospects, RAF, Unemployment, Want.*

Enjoyment Words – *Enjoyable, Enjoyment, Enthusiasm, Exciting, Fascinating, Fascination, Fun, Like, Stimulating.*

Learning Words – *Curiosity, Curious, Educated, Education, Enlightenment, Interest, Interesting, Knowledgeable, Knowledge, Learning, Scholarship, Understanding.*

Passage Words – *ALevel, Continuation, Fate, Inevitable, Parents, Progression.*

Programme Words – *Alternativeness, Change, Choice, Combination, Computers, Consolidation, Creativity, Easy, Einstein, Essential, Gadgets, Good, Hobby, Important, Logical, Maths, Programming, Relevant, Skills, Technology, Useful, Variety, Versatility.*

University Words – *Beer, Best, Independence, Leeds, Popular, University.*

Don't Know Words – *Boredom, Confusion, Date, Fish, Love, Madness, Stupidity, Wheee.*

Motivation for Programming

The analysis for the question on the same questionnaire specific to programming produced the following categories. This was also summarised in section 4.2.2.

Career Words – *Ambition, Career, Contribute, CV, Employability, Future, Investment, Job, Money, Opportunities, Professionalism, Prospects.*

Content Words – *Abilities, Compatibility, Different, Essential, Foundation, Hard, Important, Inclusive, Insight, Java, Necessary, Necessity, Need, OOP, Practicality, Relevant, Skills, Software, Specialisation, Standards, Useful.*

Compulsory Words – *Compulsory, Force, Forced, HaveTo, Mandatory, Must, Obligation, Required, Requirement.*

Enjoyment Words – *Exciting, Enjoy, Enjoyable, Fun, LikeIt, Passion, Satisfaction, Stimulation.*

Learning Words – *Challenge, Curiosity, Diversification, Experience, Interest, Interesting, Intrigued, Knowledge, Learning, Perfectionist, Veritas Vincit, Want.*

Don't Know Words – *Didn't, Elderberries, Foolishness, Love, None, Uninformed, Whooo.*

Attitude – Looking Back

The full lists of words for the survey halfway through the module (section 4.3.1 on page 100) are below. The first describes the students' attitude to the programming course that they had just completed.

Positive Words – *Achievement, Amazing, Best, Cool, Enjoy, Enjoyable, Enjoyed, Enthusiasm, Enthusiastic, Essential, Excellent, Fine, Friendly, Fun, Good, Great, Happy, Helpful, Interesting, Joy, Keen, Out of this World, Positive, Rewarding, Satisfaction, Satisfied, Satisfying, Useful, Wicked, Yummy.*

Easy Words – *Comfortable, Confident, Easy, Intuitive, Logical, Manageable.*

Difficult Words – *Complex, Complicated, Demanding, Difficult, Hard, Impossible, Intense, Taxing.*

Negative Words – *Arghh, Bad, Bored, Boring, Confused, Confusing, Crap, Dull, Exhausted, Exhausting, Flippant, Frustrated, Frustration, Grr, Hate, Hateful, Hatred, Infuriating, Lazy, Nasty, Nightmare, Painful, Stressed, Struggle, Struggling, Tedious, Time-Consuming, Tiresome, Tiring, Too Much Work, Trying, Unhappy, Useless, Why?.*

Neutral Words – *Alright, Basic, Bumpy, Challenging, Compulsory, Coping, Dedication, Information, New, OK, Perseverance, Reasonable, Relieved, Slow, Understandable.*

Don't Know Words – *Compromising, Developing, Indifferent, Money, Templateless, Testing, Varied.*

Attitude – Looking Forward

Finally these words were used to answer the question at the halfway point about the students' attitude to their programming course in the following semester. This was also discussed in section 4.3.1.

Positive Words – *Aggressive, Anticipation, Arseingear, Better, Cool, Curiosity, Curious, Determined, Diligent, Eager, Enthusiasm, Enthusiastic, Excited, Expectation, Friendly, Fun, Good, Happy, Helpful, Hope, Hopeful, Important, Improve, Interest, Interested, Interesting, Intrigued, Joy, Keen, Laziness, More Aggressive, Optimistic, Positive, Potential, Prepared, Smooth, Useful, Yummy.*

Easy Words – *Confident, Easier, Easy.*

Difficult Words – *Difficult, Hard, Harder, Tough.*

Negative Words – *Annoyance, Apprehension, Apprehensive, Boring, Clueless, Concerned, Confused, Confusion, Daunting, Depressed, Despair, Dread, Effort, Fear, Grr, Help!, Horrified, Mysterious, Nasty, Nervous, Nightmare, Oh God!, Scared, Scary, Shit, Trepidation, Unconfident, Undesirable, Uneasy, Unfortunate, Unnecessary, Unsure, Wary, Worried, Worry, Worrying.*

Neutral Words – *Alright, Careful, Cautious, Challenging, Compulsory, Continue, Essential, Forward, New, OK, Planning, Practice, Progress, Undeterred, Unworried, Waiting, Well.*

Don't Know Words – *Ahh, Blank, Don't Know, Holiday, Indifferent, Inevitable, No Idea, Opaque, Templateless, Undecided, Understanding, Unknown.*

Appendix C

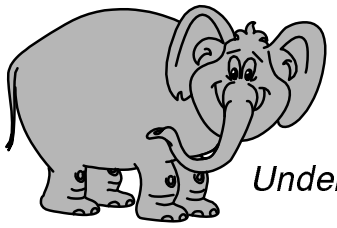
Individual Questionnaires

The following pages contain the forms and questionnaires used to investigate the individual students' experiences. This formed the basis of chapter 5.

The first questionnaire (figure C1) was used as the basis of structured interviews with the veterans. It was completed by the interviewer.

The novices were given one questionnaire before they started their course (figure C2), one every week (figure C3), and one when the course was over (figure C4). They completed the forms themselves with no guidance.

'SO11' (mentioned on the forms) is the School of Computing's code for the first semester programming module. SO12 is the second semester module.



ProgyQuoL

Understanding the First Programming Experience

Please provide the following information about yourself by circling the answer that applies:

Age:	**Under 21**	**Over 21**	
Gender:	**Female**	**Male**	
Origin:	**UK**	**EU**	**Overseas**

Looking back, what do you think about SO11 and SO12 now?

What are your general feelings about programming as an activity?

Would you say you were now a competent C++ programmer as per the stated aims of SO11?

Can you pinpoint anything that was especially good or bad about the way SO11 and SO12 were taught?

What do you think was the biggest problem for you in SO11?

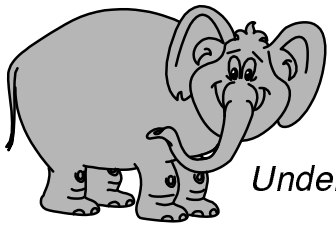
What was the biggest help?

What advice would you give people on your degree starting SO11?

Can you summarise SO11 in one word?

Form 3 - Veterans

Figure C1: Questionnaire – The Veterans' Experience



ProgyQuoL

Understanding the First Programming Experience

This project aims to understand the experience of learning to program for the first time. Programming is an unusual academic subject in that different people learn it best in very different ways. By better understanding the ways in which different types of people learn, we hope to be able to provide a better learning experience for all.

Thanks for your time. Any queries on this work can be sent to tony@comp.leeds.ac.uk.

Name:

Please provide the following information about yourself by circling the answer that applies:

Age: **Under 21** **Over 21**
Gender: **Female** **Male**

What degree programme are you studying?

Why have you chosen this degree?

And why have you chosen this University?

What do you think are the main attributes of a good computer programmer?

Do you think you have them?

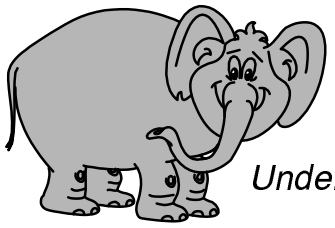
Do you expect that learning to program will be easy?

Do you expect to be good at it? Why do you think that?

*What **one word** summarises how you feel about SO11 today?*

Form 2 - Group, Before

Figure C2: Questionnaire – The Novices Before the Module



ProgyQuoL

Understanding the First Programming Experience

This project aims to understand the experience of learning to program for the first time. Programming is an unusual academic subject in that different people learn it best in very different ways. By better understanding the ways in which different types of people learn, we hope to be able to provide a better learning experience for all.

Thanks for your time. Any queries on this work can be sent to tony@comp.leeds.ac.uk.

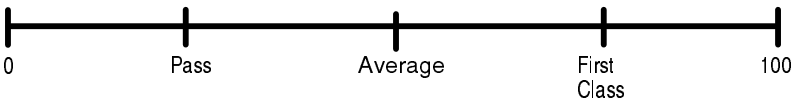
Name: Week:

*Please summarise your **experience** of SO11 this week in one word:*

Have there been any particular problems for you in SO11 this week?

Have there been any particular successes for you in SO11 this week?

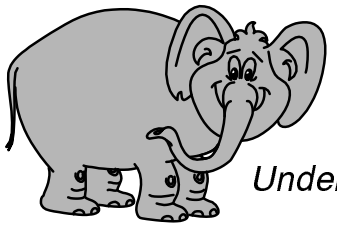
On this scale, how well do you think you will do in SO11? Draw a line.



Why?

Form 4 - Group, During

Figure C3: Questionnaire – The Novices' Weekly Experience



ProgyQuoL

Understanding the First Programming Experience

Looking back, what do you think about SO11 now?

What are your general feelings about programming as an activity?

Would you say you were now a competent C++ programmer as per the stated aims of SO11?

Can you pinpoint anything that was especially good or bad about the way SO11 was taught?

What do you think was the biggest problem for you in SO11?

What was the biggest help?

What advice would you give people on your degree starting SO11 next year?

Can you summarise SO11 in one word?

What are you expecting from SO12? Are you expecting to succeed in it?

Can you summarise your feelings about SO12 in one word?

Form 6 - Group, After

Figure C4: Questionnaire – The Novices After the Module

Appendix D

Leeds and Kent Results

The following tables show the results from the surveys described in chapter 4 with the results separated into those from Leeds and Kent. The two groups of students were treated as if they formed a single cohort throughout the main body of this study. This appendix provides the raw data which could be used to determine whether this was a justified approach. The tables are presented with some brief comments but any firm conclusions would require some detailed statistical work.

Table D1 is a complete version of table 1 on page 96. There do not appear to be any significant differences between the two sets of students except, perhaps, that only students at Leeds chose the *university* option.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
achievement	8	3.98	5	3.50	13	3.78
aspiration	84	41.79	57	38.86	141	40.99
enjoyment	12	5.97	8	5.59	20	5.81
learning	70	34.83	53	37.06	123	35.76
passage	4	1.99	1	0.70	5	1.45
programme	16	7.96	13	9.09	29	8.43
university	4	1.99	0	0.00	4	1.16
don't know	3	1.49	6	4.20	9	2.62

Table D1: Motivation for Degree

Table D2 is a complete version of table 2 on page 96. It is apparent here that a rather larger proportion of the Leeds students were viewing the module as simply a compulsory part of their course. The counter-side to this is that the Kent students appear to be somewhat more interested in learning for its own sake. Curiously there are also signs that the Kent students are more interested in the future employment and career prospects but the proportion of Leeds students choosing any category other than *compulsory* was small.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
career	10	4.98	13	9.29	23	6.74
content	29	14.43	18	12.86	47	13.78
compulsory	123	61.29	51	36.43	174	51.03
employment	6	2.99	10	7.14	16	4.69
learning	27	13.43	39	27.86	66	19.35
don't know	6	2.99	9	6.43	15	4.40

Table D2: Motivation for Programming

Table D3 is a complete version of table 3 on page 97. There are no immediately apparent significant differences here. If there is a difference it is that the students at Leeds are more likely to be studying for their own satisfaction, while those at Kent are more inclined to be interested in their future career. That said, the difference is slight and students at both institutions are almost unanimous in rejecting the other choices.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
own satisfaction	103	50.49	65	46.76	168	48.98
please family	0	0.00	1	0.72	1	0.29
please teachers	0	0.00	0	0.00	0	0.00
get a good job	94	46.08	70	50.36	164	47.81
just pass	1	0.49	0	0.00	1	0.29
don't know	6	2.94	3	2.16	9	2.62

Table D3: Before the Module – Attitude to Studies

Table D4 is a complete version of table 4 on page 103. The only significant difference seems to be that the Leeds students were a little more likely to be *negative*, while those at Kent were more likely to be *neutral*. This may be because there had been less summative assessment at Kent.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
positive	71	39.01	25	37.31	96	38.55
negative	35	19.23	9	13.43	44	17.67
easy	8	4.40	1	1.49	9	3.61
difficult	30	16.48	12	17.91	42	16.87
neutral	34	18.68	17	25.37	51	20.48
don't know	4	2.20	3	4.48	7	2.81

Table D4: Looking Back

Table D5 is a complete version of table 5 on page 103. It seems here that the Kent students are slightly more optimistic at this point in their course. Again it might be that they have at this point experienced less potentially demotivating assessment. The *expected* workloads of the two groups have been the same (or at least comparable) up to this stage so this should not have had a significant effect. The only other potentially interesting difference seems to be that a rather higher proportion of the Leeds students are focusing on the difficulty of the course – this could also be related to assessment. Very few students at either institution are reporting the course as primarily easy.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
positive	70	42.17	37	57.81	107	46.52
negative	49	29.52	13	20.31	62	26.96
easy	3	1.81	1	1.56	4	1.74
difficult	12	7.23	2	3.13	14	6.09
neutral	21	12.65	8	12.50	29	12.61
don't know	11	6.63	3	4.69	14	6.09

Table D5: Looking Forward

Table D6 is a complete version of table 6 on page 104. There are no immediately obvious differences. The trend from table D3 is maintained, though, with the Leeds students slightly more likely to choose the *own satisfaction* option and the Kent students more likely to opt for *get a good job*. There still seems to be more extrinsic motivation at Kent.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
own satisfaction	108	56.84	38	52.78	146	55.73
please family	3	1.58	1	1.39	4	1.53
please teachers	0	0.00	0	0.00	0	0.00
get a good job	68	35.79	29	40.28	97	37.02
just pass	8	4.21	3	4.17	11	4.20
don't know	3	1.58	1	1.39	4	1.53

Table D6: Halfway Through the Module – Attitude to Studies

Table D7 is a complete version of table 10 on page 110. Here the pattern first seen in table D4 is again apparent, with the Kent students more likely to be *neutral* while those at Leeds were more likely to focus on *difficult*. Of course, at this point the Leeds students had completed their assessment, while the Kent students were still awaiting the majority of theirs in the examination.

An emerging trend is that the students at both institutions are now less likely to be *positive* – the situation is the same at both. The closeness of the proportions of *don't know* responses is intriguing.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
positive	46	28.75	12	31.58	58	29.29
negative	30	18.75	6	15.79	36	18.18
easy	3	1.88	1	2.63	4	2.02
difficult	40	25.00	6	15.79	46	23.23
neutral	28	17.50	10	26.32	38	19.19
don't know	13	8.13	3	7.89	16	8.08

Table D7: Looking Back

Table D8 is a complete version of table 11 on page 110. The Kent students seem to be significantly more positive about their programming. Just under 10% of them never want to program again while the figure is some 22% at Leeds. The other figures also seem to indicate that the students at Leeds are much less confident about their programming abilities.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
"fine, I can do it"	72	43.11	24	58.54	96	46.15
"OK, but I don't enjoy it"	56	33.53	13	31.71	69	33.17
"never again"	37	22.16	4	9.76	41	19.71
don't know	2	1.20	0	0.00	2	0.96

Table D8: After the Module – Attitude to Programming

Table D9 is a complete version of table 12 on page 110. Once again the story is the same and the Kent students have a significantly more positive attitude. The different stages reached in the assessment regimes means that these figures are not directly comparable (it is possible that the Kent students' enthusiasm would wane after the examination or specifically the results of the examination) but this does hint that there may be some differences in the cohorts. One reason may be the wider range of degree programmes represented in the Leeds students and the remarkably unanimous view of the Information Systems students at Leeds (page 112).

It is also noticeable that the Kent students are very much split on this issue with precisely the same proportion choosing each of the two main options. At Leeds, on the other hand, there is a very clear majority.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
yes – would work as a programmer	59	35.33	20	48.78	79	37.98
no – wouldn't work as a programmer	105	62.87	20	48.78	125	60.10
don't know	3	1.80	1	2.44	4	1.92

Table D9: Attitude to Career

Finally table D10 is a complete version of table 13 on page 110. The trend in the first two tables in this sequence (D3 and D6) seems to have changed. The Leeds students are still more likely to have chosen *own satisfaction* but the proportion has dropped slightly from the previous 57%. At the same time the proportion of students at Kent choosing this option has increased by some 6%. Apart from this the only phenomenon of interest is that all the Kent students now have an opinion.

	Leeds		Kent		Total	
	Freq.	%	Freq.	%	Freq.	%
own satisfaction	78	46.71	23	56.10	101	48.56
please family	4	2.40	2	4.88	6	2.88
please teachers	0	0.00	0	0.00	0	0.00
get a good job	70	41.92	11	26.83	81	38.94
just pass	7	4.19	5	12.20	12	5.77
don't know	8	4.79	0	0.00	8	3.85

Table D10: Attitude to Studies

This brief look at the results from the two institutions shows that there may be some differences between the attitudes of the two groups of students. As well as the hard data, there is a general sense that the students at Kent are more positive about their experience and about programming. This may well be because they have experienced less summative assessment, or there may be other causes. It would be interesting to investigate the differences, their causes, and their effects, in more detail than has been possible here.

Appendix E

Detailed Results

The tables in this appendix show the detailed breakdown of the results from the various surveys described in chapter 4. The tables show the actual number of students choosing each option for each question on each questionnaire.

The abbreviations used for the degree programmes at Leeds are Cognitive Science (CG), Computer Science (CS), Computing (CT), and Information Systems (IS). The students are divided into the single-subject and joint-honours variants of each of the last three of these programmes. There are far fewer degree programmes at Kent. The results from Kent are split into just two groups – Computer Science (CS) and Others.

The first three tables present the results from the first questionnaire (figure A1 on page A2). Table E1 shows the students' attitude to their degree programme, table E2 shows their motivations for following a programming course in particular, and table E3 shows their general motivation for their studies.

The next three show the results from the second questionnaire (figure A2 on page A3) presented halfway through the programming course. Tables E4 and E5 show the students' attitudes to programming as they look back on the course just ended and forward to the next semester respectively. Table E6 once again shows the students' attitude to their studies and can be compared with table E3 to show how this attitude has changed.

Finally the remaining tables present the results from the final questionnaire (figure A3 on page A4). Table E7 shows the students' final view of programming and the final two tables (E8 and E9) show how they plan to approach programming in the future (if indeed they do). Table E10 completes the series of tables E3 and E6 to show the final state of the students' attitudes to their degree.

In all of these tables the *don't know* category has been used to include responses that were largely impenetrable or clearly flippant as well as more explicit choices for this option. The total numbers of responses within each questionnaire differ because of a small number of incorrectly or partially completed responses.

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
achievement	0	2	1	1	2	2	0	2	3
aspiration	0	15	23	14	8	10	14	38	19
enjoyment	3	4	1	1	1	1	1	6	2
learning	6	20	16	9	8	6	5	35	18
passage	0	0	0	2	0	1	1	1	0
programme	2	3	2	1	2	4	2	9	5
university	0	2	0	1	0	1	0	0	0
don't know	0	1	2	0	0	0	0	3	3

Table E1: Before the Module – Motivation for Degree

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
career	1	0	6	1	0	0	2	7	6
content	0	6	3	9	1	8	1	9	9
compulsory	8	28	25	16	16	12	18	36	15
employment	0	4	1	1	0	0	0	7	3
learning	0	7	10	0	4	4	2	24	15
don't know	2	0	0	2	1	1	0	7	2

Table E2: Before the Module – Motivation for Programming

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
own satisfaction	5	21	20	17	12	16	12	40	25
please family	0	0	0	0	0	0	0	1	0
please teachers	0	0	0	0	0	0	0	0	0
get a good job	5	23	25	12	7	10	12	45	25
just pass	0	0	1	0	0	0	0	0	0
don't know	1	2	0	0	3	0	0	3	0

Table E3: Before the Module – Attitude to Studies

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
positive	2	28	16	5	8	5	7	21	4
negative	3	5	9	9	1	5	3	7	2
easy	1	2	1	2	0	0	2	1	0
difficult	3	2	5	10	3	1	6	6	6
neutral	1	6	10	2	4	7	4	11	6
don't know	1	1	1	0	0	0	1	3	0

Table E4: Halfway Through the Module – Looking Back

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
positive	5	21	17	3	10	9	5	28	9
negative	6	7	10	11	2	3	10	11	2
easy	0	1	0	1	1	0	0	0	1
difficult	0	1	4	2	0	1	4	1	1
neutral	0	5	4	6	1	4	1	6	2
don't know	0	4	2	1	0	1	3	3	0

Table E5: Halfway Through the Module – Looking Forward

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
own satisfaction	4	30	21	19	10	12	11	31	7
please family	0	0	0	1	1	0	1	0	1
please teachers	0	0	0	0	0	0	0	0	0
get a good job	5	16	19	6	7	6	9	20	9
just pass	1	0	2	2	0	0	3	1	2
don't know	1	0	1	1	0	0	0	1	1

Table E6: Halfway Through the Module – Attitude to Studies

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
positive	1	14	15	1	6	7	2	10	2
negative	4	7	6	4	3	2	4	4	2
easy	0	1	1	0	0	1	0	1	0
difficult	1	1	6	12	4	6	10	3	3
neutral	0	8	11	1	3	4	1	6	4
don't know	0	2	3	4	1	0	3	2	1

Table E7: After the Module – Looking Back

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
"fine, I can do it"	2	26	21	1	9	8	5	18	6
"OK, but I don't enjoy it"	1	7	15	8	6	9	10	7	6
"never again"	3	1	6	15	2	4	6	3	1
don't know	0	0	1	1	0	0	0	0	0

Table E8: After the Module – Attitude to Programming

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
yes – would be a programmer	3	19	23	0	6	8	0	15	5
no – wouldn't be a programmer	3	15	20	25	9	13	20	13	7
don't know	0	0	0	0	2	0	1	0	1

Table E9: After the Module – Attitude to Career

	Leeds								
	Single Subject				Joint Honours			Kent	
	CG	CS	CT	IS	CS	CT	IS	CS	Others
own satisfaction	4	15	15	10	9	13	12	17	6
please family	0	0	0	1	0	1	2	2	0
please teachers	0	0	0	0	0	0	0	0	0
get a good job	2	15	24	11	5	7	6	7	4
just pass	0	3	1	2	1	0	0	2	3
don't know	0	1	3	1	2	0	1	0	0

Table E10: After the Module – Attitude to Studies