

This is a chapter contained in *Reconstructing Professionalism in University Teaching*, edited by Melanie Walker, published in the SRHE series of the Open University Press, (2001), pp105-128, ISBN 0-335-20816-9. From the book cover: "This book is the story of a higher education project [known humorously as the *Barcelona Group*], and central to the story are the attempts of university teachers to enact a critical professionalism in their everyday lives in teaching and learning; and also their development of a shared and collaborative dialogue. Each of the team [numbering six] seeks not only to improve their practice of teaching but also to explore amongst themselves what kind of professional they want to be and how to realize it in their work with students."

Engaging a Large First Year Class

Quintin Cutts

Department of Computing Science, University of Glasgow, Scotland.

Aside

As I sit to write this chapter, I have just leafed through a draft of Alison's chapter. Nightmare of comparison. Witty, clever, articulate, deep, scholarly – any number of judgments can I make against my own rather scientific writing. And yet, as my stomach settles, I realize that my whole narrative is intimately related to this one reaction. For myself, my students, and perhaps academia as a whole, I am looking for increased personal responsibility – an awakening to acceptance of self, and the consequent energy that flows when we choose and create from such a standpoint.

A 'constellation of fragility'

Academics for me seem to be living in fear of the whims of their paymasters, or of one another:

Depressing conformity has infected the normally bolshie higher education community. What is missing is the spirit of challenge, of argument for argument's sake. Have we all given up and surrendered our intellectual vigour, independence of thought or potential challenge to authority?

(Peter Knight *Times Higher Education Supplement* 1st January 1999)

My students conform to a system of which they clearly disapprove. On the topic of excessive and misguided time spent sitting in front of computers, my Level 3 students commented:

T: If the whole lab is sitting up to like twelve o'clock at night, you're not going to turn round and say but it's only twenty percent [of the final mark], I'm going home at five.

S: We can't not do it, know what I mean, because everyone else around you is doing that.

B: It's a peer pressure thing.

and most significantly for a Glasgow student:

J: And going to the lab after I go to the Celtic game... I went to the lab after I went to Parkhead¹...

(Focus group, February 2000)

All of the above speaks to me of a succumbing to our fears of what will happen if we don't live up to the expectations of others: for me about how I would be judged by others; for academics about how rebelliousness would be viewed; and for the students about how they would fare if they voiced their honest opinions about the teaching and learning environment and this early induction into a 'long hours' working culture. In short, we appear to be unwilling to take a stand for our own views on education.

In this chapter, I am aiming to illuminate my view of the inherent *messiness* of education, for both providers and receivers, a view shared by the Barcelona group. Messiness for me relates strongly to Barnett's (2000) 'constellation of fragility', constituting challengeability, contestability, uncertainty and unpredictability as the fundamentals from which we can begin to define modern university education. Messiness requires university educators to accept that there is no formula for the *right* delivery or receipt of an education. We are called to recognize ourselves as agents of change for the students, in every fresh instant, rather than mere receptacles for and dispensers of knowledge. In accepting this agency, we ourselves must be willing to be educated, to recognize and embrace the need for change, and to take a stand on and argue for what is fundamentally true for us. Education can never be neatly packaged; it is fundamentally unruly!

At the same time as highlighting messiness amongst students, the chapter presents my own messy evolution as a university academic, from a starting point as a product of the scientific research 'machine', to the position I now find myself in where I am optimistically questioning many of the ground rules by which most academics seem to operate. Crucially for me, this position is both within the system, and yet critically reflective upon it.

The chapter is based around the research project that I took on as a result of joining the Barcelona group – the analysis of a first-year computer programming module with an intake of around 450 students of varying ability, for which I am the module coordinator. The messiness of education is embodied in a series of dilemmas facing both staff and

¹ Celtic is one of the football clubs in Glasgow, and Parkhead is the name of its stadium

students when considering module content, communication and prevailing attitudes, and upon which we are called to adopt a position and to act from it:

- **Innate ability vs. teaching and learning techniques.** Among computer scientists, the belief has long been held that the ability to write computer programs is largely innate rather than learned. Against this, the more experience students have of practicing their program writing skills, the more successful they generally are. The opportunity to practice in a supportive environment, however, is expensive to provide. My belief is that programming is a learned ability, and so the dilemma for me here is whether to take a professional and ethical stand for students and against the prevailing attitudes within the discipline, or simply to accept standard practice. In taking the former position, I am called to develop a sound and graded teaching methodology that will deliver improved results, but that is also tenable with increasingly large class sizes.
- **Content vs. engagement.** While teaching this course, there have been many moments when the students are fully engaged, most often when the application of programming to the students' world is evident. However, in my view, learning the nuts and bolts of programming requires significant and less obviously-relevant study for success. The dilemma here involves finding the balance between using techniques that include all-comers to the module, and ensuring that sufficient core subject material is covered. A related dilemma is the balance between subject content and more generic skills. Students are called to decide whether to engage with the subject; staff, to decide how to encourage this engagement.
- **Safety vs. sharing.** The development of a community of learners requires a commitment to open communication between and among students and staff, permitting all parties to express opinions, desires and fears. A lack of awareness of the need for this commitment or a fear of exposure, however, often prevents this kind of communication from taking place. One dilemma here is again around resourcing – balancing investment in the encouragement of dialogue with investment required to learn the subject matter. A second dilemma for both staff and students is whether to cross the safety barrier in order to deepen learning.

I have presented the dilemmas in the order in which I encountered them while evaluating and adjusting the programming course. My initial concerns were largely technical around the subject matter. Subsequently, the difficulties of gaining and retaining motivation became clear. Most recently, I have realized that neither of the earlier stages is really of very much use unless honest, open communication between all parties is possible. This communication starts with each one of us being as honest as it is possible for us to be with ourselves.

The Module Coordinator

Before I continue, then, some truths about me. I am a scientist by training, unaccustomed to writing in this style. Indeed the very idea of placing myself in my writing is anathema to that training. Nevertheless, I am also the fifth child of a general medical practitioner

and a general practitioner's daughter, brought up instinctively to view people as individually important, to read, and to argue, usually simply for the sake of a good argument.

My family background lies in stark contrast to my education and career. I was restricted at the age of thirteen to O-levels consisting of languages and sciences only, and at fifteen to all science A-levels. These subjects tended towards a right/wrong concept of knowledge, to *certain* knowledge. The spirit of argument and discussion was absent from much of my education, which after a very competitive primary school start, was for me more about scoring highly than about the joy of learning. The end result was what mattered, not the on-going process, hardly surprising since that process consisted more of straightforward knowledge acquisition than the messy business of learning alluded to earlier which encourages self to stand by individually constructed views of the world.

My undergraduate education in the sciences did little to change this view. I clearly remember one of my professors berating the class for not rioting in the streets or at least demonstrating against university cuts, as he had done in the sixties. Yet the teaching and assessment styles of the university were as much the cause of our apathy as was the cultural training we had received to date. Whilst our training in the science of computations was as good as any other university, our training in critical and reflective thinking was extremely limited.

For me, this limited training is a key failing in the current science education framework. The following quote from Neils Bohr, (quoted Smith 1996:33) epitomizes the issue:

There are two kinds of ideas in our universe and they are represented by two kinds of statements: those whose opposites are obviously false and those whose opposites are obviously true. The first form the basis of most publications and are intrinsically unimportant. The second can point to truth and must be cherished.

In my research training, the game of certain knowledge, of gathering these first kind of ideas, was played to some extent. However, for me we also examined areas that were more about challenging our curiosity than about the necessary delivery of academic publications. "As long as there are fresh new ideas to work with every morning" was a maxim by which I lived.

My first attempt at formal university teaching, whilst still a post-doctoral fellow, was based largely on the models I had experienced as a student. The course in question was offered for political reasons, to placate various members of the department while at the same time allowing another course, to which they objected, to remain in place. The aims and objectives of the new course, set by the department, were hopelessly ambitious. I remember expressing my astonishment to a colleague at how little of the material was being absorbed by the students. A summary of his response was "You'll get used to it", a depressing induction into the low expectations that many academics have of their students.

As a full-time lecturer at Glasgow University, early advice from many quarters pointed at teaching as a dead-end activity, with research grants and research students being the path to promotion². The content of lecturer training courses was viewed by new lecturers such as Alison and myself as both useful and interesting. We were generally aware that existing teaching methods inspired neither ourselves nor the students and yet, because of a perceived requirement to be highly active in the research arena for progression, most of us knew that we did not have enough time in our everyday teaching to use the ideas proposed in the courses. Of concern here is the ability of academic institutions to define, beyond vague generalities, the characteristics and behaviours of excellent teachers. With such a definition in place, it would be possible to reward a teacher who chose to be highly active in the education arena, reusing and creating innovative teaching methods.

Conscious attempts to reflect on my teaching were confined primarily to lunchtime sessions with Alison and one or two other colleagues, where we considered amongst other things, the potential similarities between spoken and computer language teaching. In an unfocussed manner, I was beginning to realize my frustration at a teaching and learning regime that was largely a waste of our time, most particularly because we did not appear to be serving the students at all well. Why use valuable lecturing time conveying information that could be transmitted using a host of other techniques? At last, the values involving respect for individuals, inspired by my family background, were consciously contradicting the values of my scientific training.

Late in 1997, I joined a course run by Vicki Gunn and Melanie Walker of the Teaching and Learning Service that focussed on 'reflective practice'. This was my first introduction to any significant literature on education research. The course and the literature awoke me to the possibility of self-acceptance for my student-centred views.

A report I wrote for this course on a critical incident helped me to start unpacking the way I viewed myself as an educator. Reflecting on the incident, I wrote

I can see that I am becoming very involved in what it is that the students like, perhaps losing rather what it is that is best from a teaching/learning point of view. This is a common problem – people should like me, rather than my taking a stand on what I believe is correct/appropriate. Additionally, I take the views of those who disapprove more strongly than I do of those who approve.

This speaks to me of a lack of clearly defined purposes around my teaching, other than that the students should like their lecturer. It is also clearly a sign of someone who is easily swayed by the expectations of others. And yet, it is a surprise to me how many colleagues make similar comments about the effects upon themselves of disengaged students.

² This is an attitude that is slowly changing as the university policy changes.

Returning to my development, as I started the Barcelona secondment I was awakening to my concerns around student learning and about the importance of acknowledging each student individually. My own position as a professional in academia was still barely examined. The following sections discuss the module under study, students, subject matter and teaching techniques in detail. Towards the end of the chapter, I will return to my own professional development in the light of the secondment.

Module basics

The Introductory Programming module (IP1) is the first module that students encounter in any of the Computer Science related degree courses at Glasgow University. Additionally, as part of the Scottish degree structure, any student admitted to the University may take the module, even if they do not wish to continue on to a Computing Science degree. The lack of specific entry requirement results in a very wide range of abilities amongst the students on the course, from those who have no programming experience at all, to those who have written programs both at home, school, or even work for many years.

The class size has grown from around 300 to over 450 during my four years as module coordinator. The module runs for 12 weeks. There are 24 one-hour lectures given at the rate of 2 per week, alongside 6 one-hour tutorials and 6 two-hour laboratory sessions running in alternate weeks. A team of around 20 tutors staff the tutorial and laboratory sessions. For those sessions, the class is divided into groups of around 20 students each.

These are the bare facts about the module. Understanding more about the students, the material to be studied, and how to make use of the resources at hand is the focus of the following sections. Methods used to gather data about the students and their progress with the module included focus group sessions, staff-student meetings, e-mail exchanges, and discussions with tutors.

The students

I will start with the students themselves. The vast majority of the students in the module, over 97%, are school leavers who have just started university – the first IP1 lecture is on their first day of formal lectures. The disruption to learning behaviour due to the school/university transition inevitably takes months or even years to be fully overcome. Four weeks into the course, Andy had this to say about his Higher³ course compared to the IP1 course:

There was ten of us [at school] with one teacher and we're getting really close attention ... a machine each available all the time. You could do whatever you want whenever you want and you're having to adjust to

³ The Scottish education system has an exam classification known as *Higher*, used to examine students on study from the age of 16 to 17. Students usually take around five Highers and may enter university on the basis of these qualifications at the age of 17.

three hundred punters with one lecturer. I think university is really impersonal whereas school is quite personal.

(Focus group, November 1998)

On top of the academic transition, many students are away from home for the first time, although this effect may be less strong compared to other institutions, since Glasgow University has a high proportion of home students. Fees and loans inevitably affect students' conditions as well; one of the students was working for over 50 hours per week outside the University to support himself.

The majority of the students have some previous experience of programming, varying widely from those who have used computers and even written programs from the age of four or five to those who have taken introductory school classes only. Despite Andy's earlier comments, he and others were ambivalent about the teaching of programming at school. For example:

Andy: You got held back in Higher. I think probably throughout, teachers are marking the marking scheme. It is treated as a science and not an art.

John: Definitely. It's so bureaucratic ... if there's more than one equally good way to do things, as you've shown us, at school we would only have had the option to do it one way and if you'd done it the other way then we'd have it given back with red ink all over it.

(Focus group, November 1998)

Analysis of student progress at University also suggests that more than simply previous experience is at play. Of particular interest is the group of students who start the module with significant experience of programming. Some of them gain from the module and yet a proportion fall back. The following comments are from students with very similar previous experience levels, and yet their very different attitudes to the learning process are likely to affect their progress significantly.

Nigel: I think if anyone's done a significant amount of programming, they probably realize that they're going to have to change the way they do things here. You know if [the programs] get any bigger then [school techniques] just won't work.

Peter: I thought the programming bit would seem to me to be going a bit slow, but to be completely honest I find it painful going.

(Focus group, November 1998)

Nigel can see that although he is an experienced programmer for a particular class of relatively simple problems, the university course is ultimately aimed at problems of a larger magnitude and so he is willing to learn the new techniques on offer. Peter, on the other hand, appears to be unaware that there is new material for him in the course, instead falling back on his existing techniques to solve the early, simple, problems of the course.

There is an indication here that the attitude and previous experience of students will shape their ability to access new knowledge and their success on the course. This will be examined in more detail in the final section of the chapter. For the moment, the subject matter itself will be the focus of examination, in order to tease out some of the skills required to master it, and to show that these are, to a large extent, generic skills. The conflict within me between training in computer programming and university education in the wider sense becomes apparent in these two sections. My language in this next section has veered back towards a scientific presentation, in stark contrast to that of the final section on student and staff attitudes. The gap across to that section is bridged with a middle section discussing teaching methods, in which the human aspects inherent in education cannot fail to become apparent.

Breaking down the skill of computer programming

Even before the Barcelona group started to meet formally, I had worked with Judy on the relationships between her teaching of mathematics to engineers, reported in Chapter XX, and my teaching of programming. There are many similarities.

- Both classes contain non-specialists – the mathematics class consists of engineers, while around half of my class does not intend to continue with computing.
- Both subjects involve solving problems
- The problems are solved using combinations of a set of *tools*: in Judy's case, mathematical formulae, equations, proofs; in programming, the set of computational concepts that can be expressed using a programming language.
- Use of the tools is expressed using *languages*, either formulae, equations and algebra in mathematics, or a programming language in my subject.

The implication of these similarities suggested to us that we may first, be able to share teaching methods across mathematics and computer science in the areas of student motivation and problem solving and second, be able to adopt existing teaching methods more commonly associated with modern language teaching.

In our discussions, Judy and I searched for the individual components of these disciplines as a starting point to better understanding of what was happening in our classes. In particular, we wanted to determine whether the ability to program was largely innate, or whether we could find a simple enough set of steps that anyone could learn. We examined problem solving and language/tool use independently.

In his seminal book for mathematics educators, *How To Solve It*, Polya (1957) defines key steps for problem solving, familiar to most of us from our everyday experience of the process:

1. Understand the problem.
2. Construct a plan for solving the problem.
3. Carry out the plan.
4. Examine the solution obtained.

In programming, the problem solving aspect is usually thought of as consisting of only steps 1 and 2 of Polya's scheme. Step 3, *Carry out the plan*, is the translation of the plan into formal programming language code and its subsequent execution on a machine. These stages are shown in Figure 1.

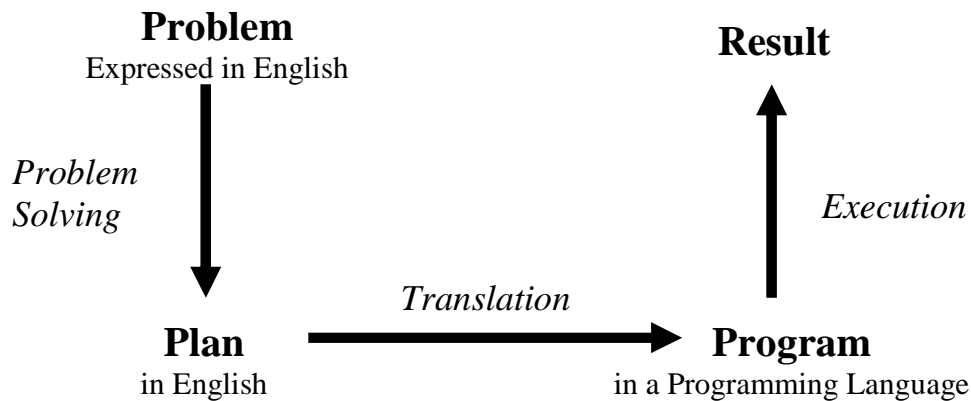


Figure 1: The Programming Process

Problem Solving

Understanding a problem fully, or grasping its essence, requires questions to be asked about any uncertainties or ambiguities in the original problem statement. A problem solver must be able to formulate such questions and be willing to ask them.

The next step is to construct a strategy for solving the problem, probably the hardest stage and the least well-understood. Hence a possible reason why the ability to find solutions is often thought of as innate. In essence, it involves designing a set of steps that when carried out will produce the desired solution to the problem. How are these steps determined?

An understanding of the available tools, and how they may be used, is required. In programming, some tools are built into the programming language, and some are fragments of code written by another programmer and made available for reuse. Compare this with a chef, who may use some raw ingredients and some processed foods, a wooden spoon and some complex machinery in order to 'solve' a particular culinary problem. An understanding of the way in which tools may be combined is required. Tools may be used in various ways – in a sequence, as an optional action, or repeated many times. These methods of combining tools are regarded as fundamental concepts of computation.

An ability to visualize the program's behaviour during execution is necessary. The plan that embodies the strategy and the corresponding program are static descriptions of a dynamic process. When the program is executed, data values are created and the steps of the program are followed, causing the data to be changed over time. Most programs will create tens or hundreds of data values that are adjusted in complex ways to follow the real

world situation they are attempting to represent. Writing plans and programs requires us to be able to breathe life into these static forms, to see in our mind's eye how the world embodied by the data values inside the computer will be changed as successive program statements are executed.

Crucially, previous experience of solving problems comes into play. When a new problem is faced, the solver may be in one of three positions.

- Exactly the same problem has been seen before. In this case, the previous solution may be reused – perhaps with slight improvements in the light of the previous results.
- No problems like this have been seen before, in which case a strategy for solving the problem must be found. There are various techniques known about for finding strategies – a common one taught in programming is *stepwise refinement*, where the problem is *analysed*, or refined, into component pieces, each of which is considered to be a problem to be solved in its own right. This refinement is continued until the components are small enough to be easily solvable, or fall into the first or third categories described here. Once the smaller components are solved, they are *synthesized* back into a single complete solution. Analysis and synthesis are common features of all problem solving techniques. One of the hardest aspects to master here is the decomposition of the tasks into pieces that can be solved with the tools at hand. A strategy for fixing a car that used a spanner would only be any good if we actually had a spanner, and knew how to operate it.
- A problem *similar* to this has been seen before. This is the most interesting position of the three. Experienced problem solvers have a repertoire of previously solved problems. Fast solving comes from an ability to *abstract out* the essential similarities between a new and a previously solved problem and so pick out the reusable parts of the previous solution. These parts constitute a reusable template or *pattern*, which may be specialized to the new problem. For example, if I have previously solved the problem of playing a game of patience, I will be aware of the strategy of repeatedly taking a move until a winning situation is reached.

From this discussion, it appears that the skills required of a successful problem solver are largely not specific to programming, and consist of the following:

- an ability to question, to be inquisitive
- a thorough understanding of the set of tools available for problem solving, and how to combine them
- an ability to visualize complex models and changes to them over time
- analysis and synthesis skills
- an ability to abstract out key features from a description to form a template, and then to specialize that template for use in other situations

Problem solving skills alone do not appear to be sufficient for successful programming however. In addition, an ability to understand and manipulate languages is also required.

Using a programming language – translation

The plan developed using the problem solving process can be expressed informally, in steps written in English, before being translated into the syntax of the programming language. The intention is that the plan is written at a similar semantic level to that which can be expressed in the programming language, and so translation is straightforward provided the syntax of the programming language is well known. However, for novices, the vocabulary and grammatical structuring of a programming language, as well as the precise semantics of the language's phrases, are rarely well understood.

Alison and I considered the relationship between modern language and programming language teaching. We found interesting correlations by considering *communicative* and *structural* teaching methods. In brief, a communicative style encourages swift application of the language to our lives, particularly listening and speaking, at the expense of perfect understanding of the language's grammatical structure. The underlying objective is to practice communicating with the language as soon as possible, to increase fluency and creativity. By contrast, the structural view encourages deeper understanding of the underlying mechanics of the language, which in the long run will ensure that translated sentences have the right 'feel' in the target language. In the short term however, the structural view is epitomized by teaching and drilling grammar, with scant attention to the real world concepts being conveyed in the process.

Programming languages differ from spoken languages in that the computer requires exact adherence of a program to the language's syntactic and semantic rules. There can be no sloppiness – the program will only run once all errors have been removed. Anyone learning to program twenty or more years ago knew no other way of using a computer and accepted that learning to use a programming language would be largely a structural matter.

However, advances in recent years have brought us *wizards*⁴ of all descriptions, the world-wide-web, and the HTML language, all of which encourage us to think that the computer will sort out any inaccuracies for us. For example, in the HTML language used to describe web-page layout, a web browser will make the best interpretation of the HTML code it can. Usually something reasonably close to the author's intention is displayed.

The relationship with spoken languages is that wizards and HTML encourage a communicative style of learning. We look at what someone else has done, make a few adjustments to match our own requirements, and hope or test that it works. Frequently the full underlying semantics are poorly understood. This may be acceptable for web pages. However, when writing software to control aeroplanes or power stations, such an approach is entirely unacceptable, and no mainstream programming languages follow it.

⁴ A tool provided in a software package to assist the user to perform some operation. The wizard requests key information from the user about the operation and then performs it without further intervention.

Overall, the case is neither for nor against either method. Rather, both are required. For example, correcting errors and maintenance of a program over time depends on being able to read and thoroughly understand the precise meaning of each statement of existing code, while at the same time being able quickly to get a feel for what large chunks of program code are intended to do with respect to the entire application. Gaining these skills requires a mix of both educational methods.

Requirements for understanding and writing programming language code include

- understanding/appreciation of grammars in general
- ability to pick out grammatical structures in a program
- acceptance that the computer cannot understand incorrect code, contradicting existing views formed using wizards and HTML
- ability to map the language structures onto their underlying semantics

I hope that I have highlighted the many aspects involved in the subject, both in problem solving and language manipulation. Personally, I think that there are a number of skills to master and that collectively this makes for a challenging subject. Rather than saying that programming ability depends on some innate quality of the practitioner, a teaching and learning strategy must cover each of these skills.

Development of teaching techniques

My analysis of the subject demands was mostly complete by the start of the 98-99 teaching session, and had highlighted a number of crucial content components required in the module. The focus of this section is the teaching methods used to convey these components in a manner which aimed to engage the students.

To reiterate, the aim is to provide a module that is of value to *all* registered students – those with no, and those with significant, previous knowledge. Additionally, a secondary aim has been to maintain the primary aims and objectives of the module, and the module structure, set as it is within an existing degree and staffing infrastructure. These two aims have created a number of dilemmas in choosing appropriate teaching methods.

Practical subject vs. lecture-based module

Programming is a skill and therefore requires repeated practice for mastery to be achieved. As a successful student put it, 'Lectures are good, self-study sheets are helpful to see what you are after, and the laboratory is what it is all about.' (e-mail, February 2000). By this, the student meant that the time actually writing and playing with programs was the most beneficial time. Supervised laboratory time for large classes is expensive to supply however, whilst lecturing to a large group is relatively cheap. The challenge for me has been how to maximize the limited laboratory time available to the students. I have addressed this in two ways.

In the first session I researched (1998/99), I broke each lecture into periods of teaching and of practical exercises. In the latter, the students were encouraged to practice what

they had just seen and heard on a small exercise. As well as helping to embed the material for the students, this ideally gives me a mid-lecture opportunity if most students are having difficulties. In practice, I found this latter aspect hard to realize, as the layout of a 300-seat theatre prevented personal contact with the majority of the class in order to assess their development. Nonetheless, many students found the active style of the lecture beneficial, and made these sorts of comments:

The interactive approach was excellent, helped to keep you focused and concentration levels up.

Getting people involved by putting up programs to hand-execute was good and brought forward any problems with understanding.

(Course questionnaire, January 2000)

On the other hand, I saw many students not attempting to engage in any way in the exercise sections. Student attitudes to learning, discussed in more detail in the final part of the chapter, are strongly in play here. As one student said, 'In-lecture work was not very good because no one concentrates on work and would rather wait for the answer. Good if you concentrate.' (Course questionnaire, January 2000)

As well as active participation in lectures, I am also keen for students to make effective use of their own time. Observations of tutorial and laboratory performance indicated to me that many students had difficulty bridging the gap from relatively passive lecture material to active engagement in practical work, either on their own or at a machine. To counter this, in the second teaching session (1999/2000) I introduced a series of guided exercise sheets, one for each lecture. Known as self-study sheets, these were an optional extra for the students, intended particularly to assist those students with little or no programming experience. They consisted of relatively easy material similar to the exercises used in lectures, along with fully worked-up answers.

Reading student feedback on where the students directed their energy, I can see a real confusion about the role of the self-study sheets, and how to make best use of them in the context of course components viewed as mandatory, such as the assessed laboratory exercises. The strongest link that was *not* made during the course by students was that completion of self-study sheets would reduce the overall time required on the assessed work. Here are a number of responses of different kinds.

Most of my time spent on assessed laboratories, I did not do enough SSS (Self-Study Sheets)

Tended to ignore SSS, didn't feel they were of any worth. Not sure why. Probably due to previous experience i.e. done it before. [SSS not really intended for a student of this level]

SSS were useful and didn't take too much time [from a beginner]

SSS useful, but get neglected when more pressing things in other subjects arise i.e. coursework, exam revision – can lead to difficulty in laboratory work.

I found that about 90% of my time was spent on laboratories. I never did the SSS as I felt other things I had to do were more important and could be more beneficial. The fact that marks in the laboratory were what would help me pass the course, I concentrated heavily on these.

(Course questionnaire, January 2000)

This concentration on marks is discussed below. Overall however, the comments on the self-study sheets and lecture exercises were positive. The following comment from one student indicates that they encouraged productive use of a student's time.

Since I had no programming experience before, I'm happy to have a little knowledge now. But I think it's not much. But I know the things covered by IP1. In Germany I don't normally know the things after a lecture. I need much more time to repeat and repeat... I liked IP1 because this repetition was covered by the lecture (and SSS, lab).

(Course questionnaire, January 2000)

My reading of this response is that the student was able to keep up with the course as it progressed, an essential requirement in programming, where each new piece of knowledge builds on what has gone before.

Problem solving vs. language learning vs. concept learning

Initial analysis of the course with staff in the department indicated that problem solving was the aspect that students found most difficult. I addressed this issue in the first teaching session under study by spending significant time on this aspect early in the course. I had made two assumptions: that the basic tools or concepts of computation, such as repetition, sequence and state, were relatively easy to absorb; and that translation from a plan to a program was easy. These assumptions were shown to be incorrect during the course. One student said

I think for people who have had no experience whatsoever, one of the main difficulties is going to be coming to terms with another language altogether, because it was a bit of a shock to me. Learning that there is a new language and how to use that new language to get what you want, that's going to be quite difficult.

(Focus group, November 1998)

A mid-term surprise test asking the students to show the outcome of executing a simple program demonstrated very clearly that many of them had not grasped the basic concepts behind the language. Interestingly, many students who stated that they had 'reasonable' programming experience fell into this category.

Nonetheless, the staged planning of a problem solution is still a large hurdle for beginners and experienced students alike. A student with Higher and SYS⁵ Computing qualifications said in a focus group discussion with Melanie and myself:

N: The most difficult part about it, the tutorial questions and the exam and stuff is that you are expected to actually put a plan down and I just wasn't used to doing that. I was just used to, because of what I'd done in school, I was just going straight to the computer and just going, like ad lib again basically, so my mind is more tuned into sort of looking at a problem and then going straight to code rather than going, sort of pseudo English but I'm getting better at that now. So it's easy to do that for like small problems but it's not very easy to do it for bigger ones unless you actually, sort of break it up into procedures and then you can do it. That was the hardest part I found with the tutorial questions, that you quite often, you were supposed to say right, why did you do it that way rather than this way and to me it was just kind of instinctive.

(Focus group, April 1999)

At issue is that the student does the task, but does not know why or how they have succeeded – they have no explicit structural rules on which to draw the next time. Yet, elucidating the process is of key importance for successfully working in teams of designers and programmers, and an essential end skill of the degree program, if not of this module.

In the second teaching session (1999/2000), on the basis of observation and feedback, I introduced the computational concepts first by getting the students to concentrate on examining small programs and making changes to them to thoroughly embed understanding of the concepts. Later in the course, I introduced problem solving, believing that the building blocks for programming were in place. The early part of the course was well received, but then the step up required for solving problems came as a surprise to students. The early part was not in fact thoroughly embedded. Responses from several students echo the following e-mail from a student:

I think when we were faced with it [the first major problem solving exercise], it came as quite a shock as the lectures hadn't really made it clear that we would be expected to write programs like this. I am sure that many beginner programmers who had gone to (and listened to) all the lectures really felt that they had missed something when the assignment was handed out.

(E-mail, January 2000)

Although I had covered it in lectures, students could not see how to link the two stages of work, resulting in a core belief in many that the course was designed for those who had

⁵ SYS is a further grading level for Scottish students who choose to stay on for an extra year after their Higher year, from the age of 17 to 18.

programmed before. Such a belief could be used by the students as an excuse for opting out, a situation discussed in more detail in the final section of the chapter. Being clearer about the course structure and expectations will also help reduce the potential for unconsciously adopting this excuse.

Motivational vs. curriculum material

The final dilemma for me in this section concerns the need to balance course content that is engaging to the students with concepts and techniques expected in future courses in the degree program. Having thoroughly analysed the requirements for programming, I am amazed at how much we expect of our students. As a result, I have been attempting to hone down the content to an essential minimum, whilst making the remaining material as relevant to the students' lives as I can. Little is learned when the connection between new and existing knowledge cannot be made. A course where the content is viewed as irrelevant is quickly classed as boring and completed only through requirement, not choice.

Traditional programming courses have centred on programs that interact with the outside world using text. A common early program might read in a name typed at the keyboard and then write out to the screen a message welcoming the person with that name. This is useful for learning programming, but hardly relevant in an age where students have WAP-enabled phones, have been playing virtual-reality video games and using highly interactive windows-based computer systems for years.

My initial attempt to tackle the obsolescence of text-based systems has been to work with graphics-based programs, effectively programs that have more in common with video games. Students can much more easily grasp the effect of their coding on the graphical output, and respond more positively. I am caught however by a need to also introduce the text-based interaction techniques for further courses. This need seriously unbalanced my course, coming in the middle section, and overloading many students.

Judy and I found we had similar concerns in this respect. Lecturers, specialists in the subject, often view a large core of material as essential for those wishing to become specialists themselves. Both Judy and I have large numbers of students who have no expectation of becoming specialists. We have been looking, therefore, at the minimum detail required to gain understanding of the fundamental subject matter, suitable for potential specialists and non-specialists alike.

The dilemma here is for me to take a stand in my department for large-scale re-organisation of the syllabus in order to accommodate the change in focus that I think is required. I am very aware of the resource cost of significant course re-tooling. And yet I am also aware that the current balance is serving our students poorly.

Reflection, questioning, safety, sharing

Bound up in the dilemmas discussed so far in this chapter is the requirement for students to be able to reflect on the processes unfolding within and around them. For students to

engage fully with course content, they need to reflect on their purpose for studying the subject and on their own learning processes. A student who is able to see how their own beliefs, fears and values interact with course material and methods is likely to be able to make necessary adjustments more quickly when the two are in conflict compared to someone who reacts automatically. Additionally, reflection by and particularly between students and staff is likely to involve participants' crossing habitual safety barriers in order to reach new understanding about self and others.

A thread running through many of the conversations within the Barcelona group has concerned students' ability to ask questions, as a means to reflect on their progress and to strengthen engagement with the material. This question-posing skill is arguably the most important learning 'skill' for students to acquire..

In one of our early meetings, I had explained to Alison my teaching methodology, which at that time involved taking a problem, analyzing its major components, writing a plan for the program and then translating that plan into a program using a programming language. Alison said:

When you've tried to get me to understand programming languages, you've put [a program] on the table and said "This is the programming language, ask questions about it". I haven't been interested in the planning and I haven't been interested in the sort of generalization and translation and all the concepts stuff and I wasn't quite sure where it was going or why it was even important, but what I'm saying is ok, well those colons are obviously there for a reason, they're in a pattern, that pattern of work is obviously there for a reason, why is it there, and then starting to ask the questions.

(Group discussion, November 1998)

On another occasion, Mike commented on a session which I recounted to the Barcelona group where I had fully engaged the students. He said, 'You were saying that there's this incredibly extraordinarily interesting set of questions we should ask, and once you've asked the questions which are fascinating, there's the small matter of how the hell, what do you need in order to answer them.' (Group discussion, January 2000)

I did in fact ask the class to write down any questions they had about the module so far, after that session. They were all at a very high level, not a single one at the level of the subject content I was currently covering in the module at that time. Examples are:

1. Why are we learning Ada and not a cross-platform programming language like Java?
2. Why does the course suddenly jump from something we can all understand to something which appears able to be done only by people with computing history? Did I miss something?
3. Can I connect my programs to colour graphics output?
4. In school you learn a language and then at uni you learn a new language, so it is a bit confusing. So why do languages differ?

5. What relevance does IP-1 have outwith the computer science course?
6. Why do we need maths?
7. What will knowing computer science give me over an ordinary Windows user?
8. Where is the internet?
9. Will I get a good job having taken this course?
10. What will be in the exam?
11. Do you ever brush your hair?

Note that with both the students and Alison, the questions relate directly to whatever fascinates the questioner – another example of teaching needing to attach to the existing hooks of knowledge in order to engage the students' interest. From the above list of questions, numbers 1, 5, 7 and 9 relate to issues around career prospects, 2, 6 and 10 to passing the module, 1, 3, 4 and 8 to the general subject area and number 11 to the lecturer. This one experiment highlights the heterogeneity of a group of students, from those who want to know what it will take to pass the exams to those keen to find out more about the lecturer himself! And yet many still hope to find a 'one size fits all' educational model.

All this opened up explicitly for me the inherent messiness of university education. Associated issues of fear, reflection and engagement are illustrated in two crucial critical incidents that occurred in the lecture theatre. The first occurred as a result of two student e-mails and some student feedback about the course so far. An e-mail from Tim suggested that 'The assignments are good, but as you keep emphasizing in lectures, since this subject is a "doing" subject, why on earth is the coursework only worth 20% of the module mark???? Surely, given the amount of time I, and others, spend on these assignment then surely it should be more like 40-50%' (E-mail, December 1999). A second e-mail, this time from David, expressed the student's frustrations: 'this whole course is a joke how the hell are u ment to go from moving a stupid wee man across a room to a complex program like sun earth moon. U just keep on adding bits to the assessment and people cant even do the bits before it. Its not fair on the people who haven't done programming before. THIS IS MENT TO BE INTRODUCTARY PROGRAMMING' (E-mail, December 1999). Finally, a third anonymous comment as part of feedback on the lecture said simply: 'Kill John Smith [one of the other Level 1 lecturers]' (Lecture feedback, December 1999).

All of these speak, in quite different styles, of confusion with the course structure, and certainly in the latter cases of extreme disengagement. The second e-mail was a whole critical incident for me, in its own right. Aware of how damaging these views could be to students' willingness to tackle the material of the course, and to wake up students to the processes and reactivity in which they were immersed, I summarized my thinking around these comments in two slides presented at the end of the next lecture, and shown here in Figures 2 and 3.

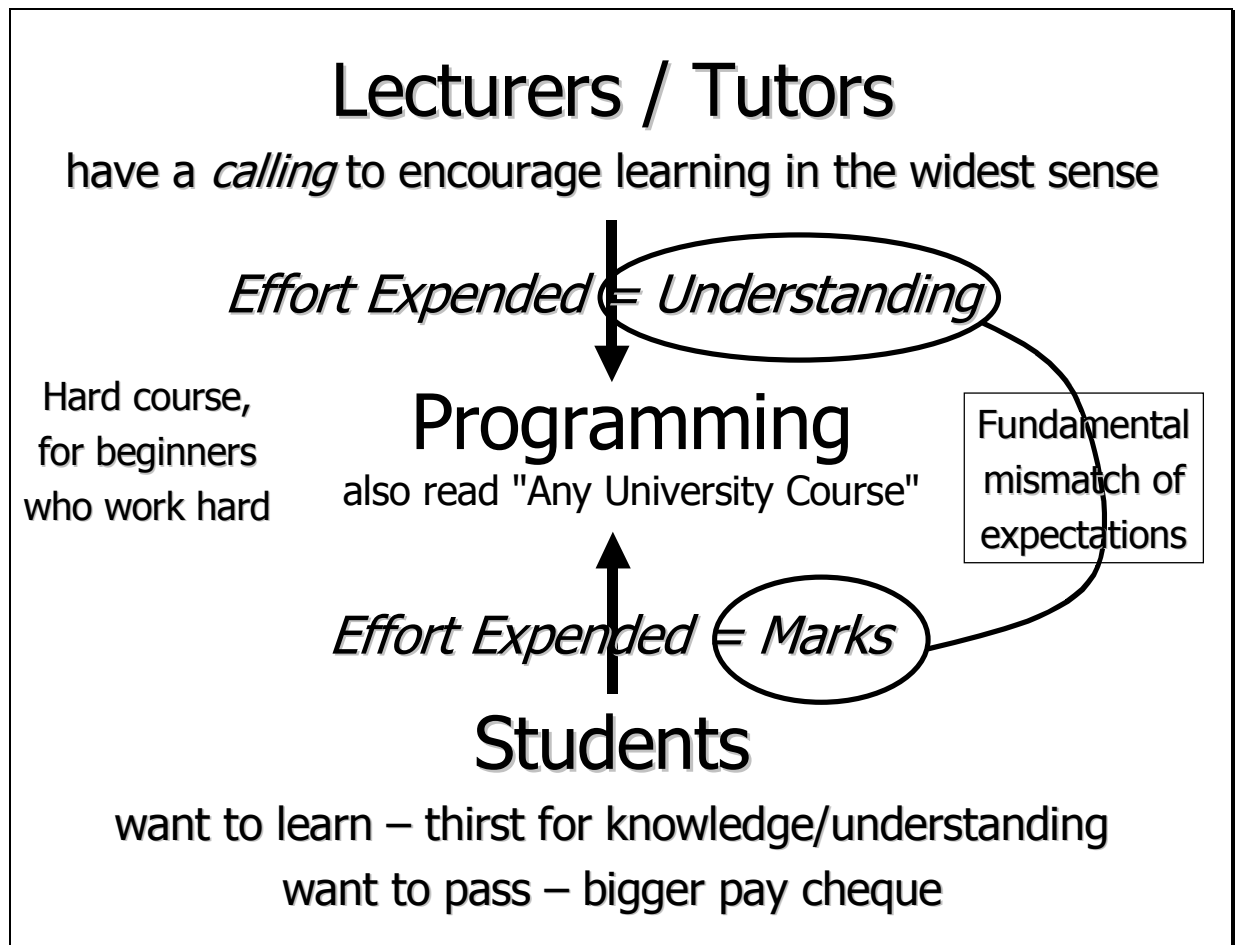


Figure 2: Student and staff attitudes to the learning process.

The slide in Figure 2 primarily answers the question posed by the first e-mail, by making clear the differing viewpoints that staff and students have of the effort expended by students during a course. In most cases, the staff are keen primarily for the students to gain understanding about a particular subject, and the fact that assessment is required at all is only as a test of that understanding. The students, however, often consider that effort expended deserves marks, and hence a pass in the course, with little connection to the appropriate understanding expected by staff. There is a mismatch of expectations here, which leads on to the second slide in Figure 3, which describes a process of communication breakdown between staff and students. The lecturer hands out a series of exercises for the students to attempt, as part of a firm belief that the only way to learn the subject is to consolidate lecture material with practical work as soon as possible. The students do not see the purpose of these materials as scaffolding to help them pass the final assessment, and increasingly view them as pointless hurdles that must be overcome to succeed in the course. The result is a resentment of the staff and the course that leaks out in disengaged behaviour: silence in tutorials, lack of preparation, and terse or rude e-mails to staff. The resentment of the students is felt by the tutors on the course and is

often returned with behaviour that cuts off those students most in need of assistance. Both sides lose out, with students becoming increasingly afraid of failure and staff simply not enjoying contact time with the students.

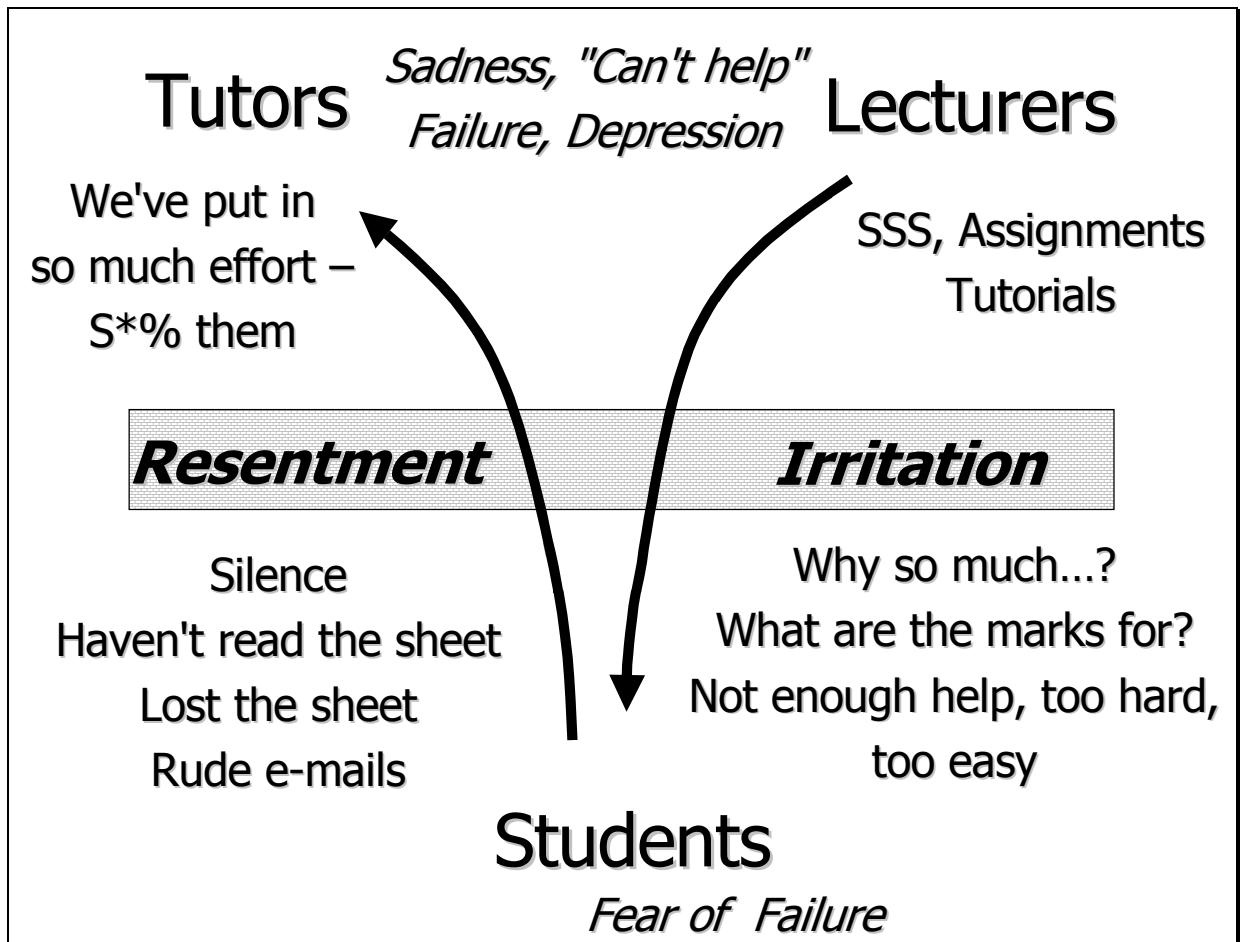


Figure 3: Communication barriers between staff and students.

When I used the personal attack on John Smith as an example of resentment/irritation, many members of the class laughed. When I stated clearly that this was not funny for me, a few still giggled. It took a repeat of this, and an enquiry as to whether they really thought this was constructive feedback, before the class as a *whole* engaged with the issue, and examined their response. As Alison commented "it's interesting to reflect on how is it you know that you've got them, what are they sending out, and it's actually silence". Indeed, I could hear a pin drop in the 300 seat lecture theatre at that moment, but in addition there was a tangible increase in energy.

The final slide of the lecture, shown in Figure 4, encouraged students to view university education in a manner most likely different from any they'd had before – exploring in an environment of uncertainty but safety under the watchful eye of the teacher – the messy view of education.

The following excerpt from a student's e-mail is typical of responses to the lecture.

C: Personally, I think that you should carry out that type of lecture, not periodically but occasionally – because it's very important to keep the link between us the students and the lecturers/tutors. I only realized this myself today when you showed the slide on what you thought was happening between the two.

(E-mail, December 1999)

Designed to make you wrestle with ideas – <i>to think</i>	<i>You</i> learn. We light the way, but do <i>not</i> teach	Encourage you to explore your strengths and weaknesses
<h2>University courses are hard</h2>		
Expect you to work independently and from your own motivation	Clean yourself up when the s%\$t hits the fan	We are not here to <i>tell</i> you how to think (or to program for that matter)
<p>"<i>too hard</i>" and "<i>too difficult</i>" are not in the University's dictionary Anything goes – either you want this, or you should be elsewhere</p> <p>You require a large dose of <i>personal responsibility</i></p>		

Figure 4: The messy view of university education.

The connection generated by this kind of interaction has been very compelling for me, and there has been a danger of valuing it above the course content to be covered. And yet at heart is my own passion to teach *all* the students this more universal material, to wake them *all* up to their reasons for being present in my lecture theatre. At a practical level, the learning for me from the incident is the crucial importance of keeping the students informed of the purposes of my teaching strategies, and to make them aware of the way that internal fears can lead to anger and resentment.

The second incident, and ensuing discussion with the Barcelona group, illuminates this passion. On another occasion, I had asked a noisy contingent of students at the rear of the theatre to be quiet, suggesting that since they had a choice to be there, no lecture being compulsory, it was unhelpful for them to sit in the theatre and talk their way through the lecture. This was not the first time that the work of the class had been impeded in this way, as I recounted in a group discussion:

Quintin: I asked them quite passionately to be quiet and then as I went down the steps I heard this giggling from the back of the theatre and I just absolutely lost my rag; I had never done that before. That was another case where my passion about “why are you here, what is this all about?” [came out.]

Mike: What was their reaction?

Quintin: Well... connection again, absolute connection like, you know, to me it seemed like I'd contacted right down to something that got them questioning “Why am I here?”

although some of them may have been plain embarrassed, of course.

Mike: All of this is attached to a need or a desire to teach something. So you don't explode because you're in a bad mood, you explode because something un-named is interfering with your ability to do the teaching business and that something which is interfering is not a technical problem, it's a much deeper one. It's about your relationship with a group of people, a diverse group of people.

Melanie: And it's inclusion into the structures of knowledge. So it's inclusion into getting a sense of being part of a community of learners.

(Group discussion, January 2000)

My over-riding sense here is of the importance of authenticity. My most committed moments with the students are when I am authentic, right to my toes. Once I had “lost the rag”, I owned my feelings in my chest at that moment, apologized for the outburst, and explained about my passion for them to be in my lecture for a worthwhile reason, to tackle their university education with that same reason. Somehow, in seeing my humanity, the students are able to engage with their own deeper purpose.

The following e-mail from a student underlines the importance of an inclusive approach. the student writes 'At today's lecture you brought up some issues that were desperately needing attention. It pisses me off when people start talking... you must have extreme will power not to shout abuse at these individuals as I know I have fallen out with a few students doing just that' (E-mail, January 2000). I am sad that I have not received any response from those at the rear of the class. Judging from the following e-mail excerpt from a student, I should guess that they are thoroughly disenfranchised from the learning and teaching process and chose to demonstrate their frustration with their non-participatory presence in the class. I expect others demonstrate it with simpler non-attendance. In passing, the student said to me,

This is on a different subject but I tried sitting in the back row a few lectures ago. They all seem to think they are rebels and they really have no cause. I wonder why they turn up in the first place. They just sneer at everything said in the lectures. There is a palpable sense of utter ignorance there, a place where I shall not be visiting again I think!

(E-mail, January 2000)

In these critical incidents, I am inspired to act out the kind of reflection that I am looking for from the students, perhaps most particularly from those in the back rows. In the first incident I received feedback, polite and rude, from both students and staff, then reflected on what was happening in my module and presented my findings back to them. I opened a channel of communication. In the second incident, I also opened a channel of communication to let the students know my real feelings in that instant. The Barcelona group commented:

Mike: You've said "Hey, I'm reflecting on what you're doing to me and this is my reaction to it" so you're enacting what it is you want the students to do but equally you're doing it in a way that is trying to include rather than divide. So you've got this diverse group, some of whom are behaving well and some of whom are behaving badly and what you're doing by turning round at that moment, is actually embracing the whole theatre and bringing them all together and including them.

Melanie: Your story, more than any of the others, is about not having the ideal group necessarily to work with. It's a very large group with very different aspirations and expectations and that's why the notion of the inclusivity of the teaching is so important because your aspiration, your value position is to try and find a way to teach all of those students.

(Group discussion, January 2000)

Conclusion

To put into perspective what programming teachers are trying to do, Alison exclaimed while comparing German courses with Level 1 programming courses, 'That's like trying to teach the whole of German grammar and a large chunk of literature in one semester!' In another conversation with colleagues, one commented that in modern language terms, programming teachers were perhaps putting together in a single course native English-speaking students who had no knowledge of foreign languages with those who had widely varying degrees of ability in some or all of French, German, Italian, and Spanish, and then teaching them all a beginners course in Greek.

As the center sections of the chapter indicate, I have been trying during the Barcelona secondment to discern a *right* way to teach programming in a short module. The truth is of course that, in line with messiness and uncertain knowledge, there is probably no right way. In each new teaching session, I match available resources with my current understanding of the discipline and the environment, and teach as best I can to every member of the class. The realization about uncertainty, about which I have told my students often, comes home once again. I take full responsibility for my actions and my

life and in the same breath release myself from the demand for some unattainable perfection. In recent years, by contrast with my early reflective writing, I notice that I care less about whether the students like me. Indeed, I find that the students write less about the qualities of their lecturer and more about what they have gained from the course itself. In the role of teacher-expert to the novice, I have been modeling both a *way of being* with the challenges around us, while at the same time teaching masterful programming, as far as I know what this means.

Whether we like the currently popular phrase *life-long learning*, we are confronted by it. Everything we do changes us and requires an openness to the new learning available. The students, whose cycle of change is the most evident, are challenged to adopt responsibility for themselves and their learning beyond that of their schooldays. My colleagues, my department, academia as a whole are all challenged to accept the changing nature of higher education and to make a stand for our fundamental beliefs. And myself? I am challenged repeatedly to learn that there is no right answer to the many dilemmas in life, and to acknowledge my ability to search for a worthwhile path nonetheless.

References:

Barnett, Ronald (2000) *Realising the university in an age of supercomplexity*. Buckingham: Open University Press.

Smith, Terry (1996) *The Ideas of the University*. Research Institute for the Humanities and Social Sciences in association with Power Publications, The University of Sydney, NSW.

Polya, G. (1957) *How to solve it*. Second edition. Princeton University Press.