Programming in the Real World:

Computer science students' perceptions of the values and difficulties of learning formal methods

Professor T. O'Shea, Dr. P. Fung
Centre for Information Technology in Information
Open University, Milton Keynes

Professor R.Bornat, Dr. S. Reeves, Dr. D. Goldson
Queen Mary andWestfield College, London

Abstract

It is generally acknowledged that computer software design would benefit from insistence on a high standard of consistency and reliability. One solution to this is the application of formal reasoning techniques to establish the correctness of programs at an early stage in the design process. An increasing number of universities are integrating this approach to programming into their undergraduate computer science courses. This raises issues related to the process of learning formal reasoning methods and the difficulties computer science students may experience on such courses. In the light of the results of a preliminary survey of students' experiences in this area we look at the factors which may contribute to their difficulties and consider the implications for computer science teaching.

Introduction

The contents of computer science courses at undergraduate level can vary greatly but on a majority of courses 'software engineering' is an integral part of computer science studies. Many graduates of these courses go on to become software engineers in industry, while others enter the academic system and eventually are themselves responsible for shaping the ideas and development of future computer scientists. It is natural that those involved in undergraduate computer science teaching should be concerned that the content and quality of the curriculum offered to students reflect the best practice and thinking in that field.

A major concern, discussed in detail by Gries [1991], is the need to develop a rigorous and scientific, as opposed to an operational, approach to programming and software design. Those supporting this view believe that if software engineering is to become a legitimate branch of 'engineering' then it must develop the same rigorous approach to design and planning that is taken for granted in traditional engineering projects. They feel that the high standard of reliability and quality demanded of sophisticated software systems today can be better satisfied if the specification and design of those systems are underpinned with verification of a mathematical rigour. Employing formal methods for program verification is seen as an essential step in achieving a precision and clarity of program design [Jones 1986]. The computer science department at Queen Mary and Westfield College, London, specifically introduces this way of thinking into the core of its first year curriculum [Bornat 19**]. Successful teaching of formal reasoning techniques and the principle which underlies their use, i.e., that programming and software design is a considered and formally provable process rather than an ad hoc 'try it and see' affair, is seen to

have long term advantages both for the study of computer science and practically for the students themselves. Gries [1991] summarises what he sees as the advantages for the latter:

> one learns that the shape of the formalization can itself lend insight into developing a solution. One acquires the urge to clarify and simplify, to seek the right notation in which to express a problem. One acquires a frame of mind that encourages precision and rigor. This frame of mind can have a strikingly beneficial effect on what work one does later as a professional in computing.

The benefits of using a formal approach to software development are indeed relatively easy to appreciate when one considers the growth of software applications designed to be used in circumstances which demand a high level of reliability. However in computer science departments it is recognized that learning to use formal methods is not without its problems for undergraduate students. In an attempt to alleviate the difficulties which students experience in this area, staff of Queen Mary and Westfield College computer science department, in collaboration with the Open University, are currently undertaking a three year research project funded by the joint research council .

In the course of the project, the collaborators expect to introduce and evaluate the use of on-line mechanical aids which have been developed to help students in applying formal rules and recording the results, thus relieving them of some of the more tedious aspects of calculation entailed in program verification. In parallel with this, they are undertaking empirical studies designed to help identify which aspects of learning formal reasoning techniques are most problematic for students.

At the outset of the project, a preliminary study was undertaken, in which current second and third year computer science students were questioned about their attitudes to learning formal reasoning techniques and the difficulties they had, if any, with these techniques during their first year courses. The study was undertaken with a view to gaining a first insight into students' perceptions of the values and the difficulties of learning formal methods. Marginally over fifty, out of the one hundred and fifty second and third year students who were contacted, completed a questionnaire. While the number of replies was obviously not large enough to be used as the basis of any generalisations, the replies themselves were interesting and useful in bringing to attention some of the factors which students see as contributing, adversely or otherwise, to their progress.

As a follow up to these questionnaires twelve students were interviewed individually and each talked in more depth about their first year experiences of learning formal methods of reasoning.

The first year courses about which the students were questioned, primarily aim to instil principles which encourage the development of 'reliable' 'correct' 'consistent' programs, all of which are qualities demanded of software in the 'real world'. Yet the answers we obtained from questionnaires and feedback from the interviews indicated that for a number of students, in their initial perceptions of computer science there exists a gap between the academic view of computing and computing 'in the real world'.

Interestingly, however, many of the students explained that they had modified their initial perceptions as their understanding of computer science had increased. Before we look at these student responses more closely and discuss the implications which they may have for computer science teaching, we give a brief description of the preliminary study itself.

An overview of courses incorporating formal reasoning methods

In the study, second and third year students were questioned about their learning experiences and attitudes in relation to certain courses they had taken in their first year. The four courses chosen for consideration were those in which learning about formal reasoning techniques formed a significant part of the work.

Two of these were taken in the first semester:

i Programming I (PI)
ii Introduction to Discrete Structures (IDS)

PI was a pre-requisite for most of the computer science courses and was based on teaching language independent concepts of programming, with emphasis on the proof of programs against informal specifications. However, mathematical experience beyond O-level (GCSE) was not required for the course.

IDS covered the basic repertoire of formal notions used in computer science, describing programming language, constructing relational databases and in developing correct programs. It covered topics such as equivalence relations and partial orders, sets, functions and relations, induction, graphs and trees and lambda notations.

The remaining two courses were taken in the second semester:

i Introduction to Logic (IL)
ii Programming II (PII)

IL introduced the basic ideas and notions of classical propositional calculus, with emphasis on the notions of proof and validity. This logic was then extended to classical predicate calculus and the notion of proof considered in more detail. The course is designed to help students realise how closely computer science and logic are linked and that mcuh of logic is inseparable from the notion of an effective algorithm [Reeves & Clarke, 1990].

PII was concerned mainly with the use of data structures and was a follow on from PI of the first semester. The programming language Pascal was taught and used to implement concepts learned in PI.

The questionnaires

The tick box format of the questionnaires required the students to rate each of the four courses in terms of how easy they had found it, how relevant to their computer studies they had found it, how interesting it had been. e.g.

Did you find this course interesting? (please tick appropriate box)

very ☐ *fairly* ☐ *not very* ☐ *not at all* ☐

Students were also invited to make additional comments on the courses, or their ratings of them, as they wished.

The interviews

The structure of each interview allowed the interviewee to discuss what was found hard about the four courses, if any difficulties had been encountered and if so, how these difficulties could be minimised. The 'interest', 'relevance' and 'easiness' aspects were a suggested framework, but interviewees were free to develop the conversation in other directions if they wished. The goal of undertaking in-depth interviews with a selection of students, was to investigate in more detail the outline picture obtained from the questionnaire ratings and to look for any commonality in the difficulties they may have encountered or the attitudes they held towards the courses.

Participants

The second and third year students of spring 1991 were circulated with questionnaires, in total about one hundred and fifty students, but the data collected refers to the sixty-three students who actively participated, i.e. fifty-one who returned completed questionnaires about the courses and twelve who volunteered to be interviewed personally. Of those who had completed questionnaires, thirty-one had given their names, while twenty of the forms were returned anonymously. It was possible, using end-of-year exam. results, to check the academic progress of the named participating students.

Overall 2nd. year results (n=79)		2nd.yr named questionnaires (15)	
good passes	31	good passes	10
comfortable	27	comfortable	4
struggling	15	struggling	1
deferred/withdrawn/transferred	6		

Overall 3rd. year degree results (n=81)		3rd.yr named questionnaires (16)	
firsts:	5	firsts:	3
upper twos:	28	upper twos:	8
lower twos:	19	lower twos:	2
thirds:	13	thirds:	2
passes:	8	pass	1
deferred/ftq:	8		

Fig. 1 Summary of end of year examination results

As previously mentioned, it is not possible to generalise from the thirty-one named students who completed questionnaires. It is, however, in light of their end of year results, reasonable to presume that comments from this set of students at least, were not unduly coloured by the experience of basically finding the whole undergraduate program very difficult. Comparing their results with the overall levels of attainment (see Fig.1 above) for their year, these participants showed a near average spread of attainment, perhaps biased towards the upper levels of achievement.

Some data from the questionnaires

Figures 2, 3 and 4 summarise students' responses. Looking briefly at each we can see that for those aspects of the courses directly queried, i.e. how easy, how interesting, how relevant a course was, some interesting points emerge.

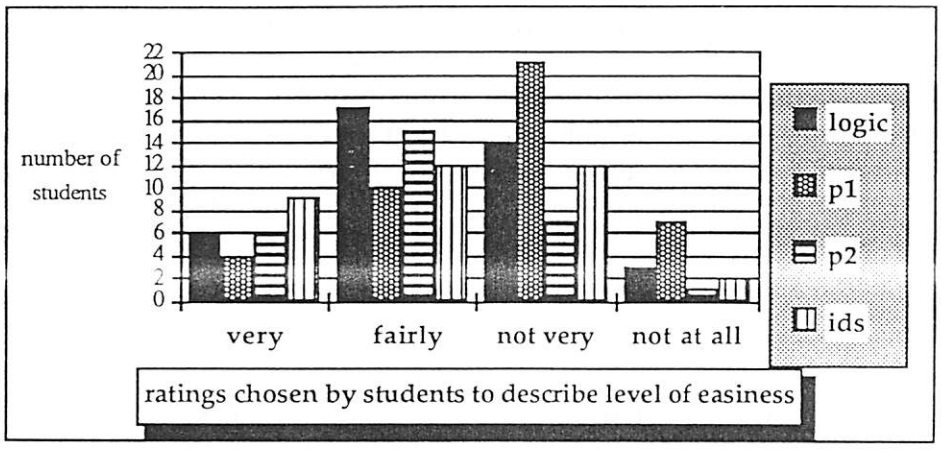

Figure 2 The easiness factor

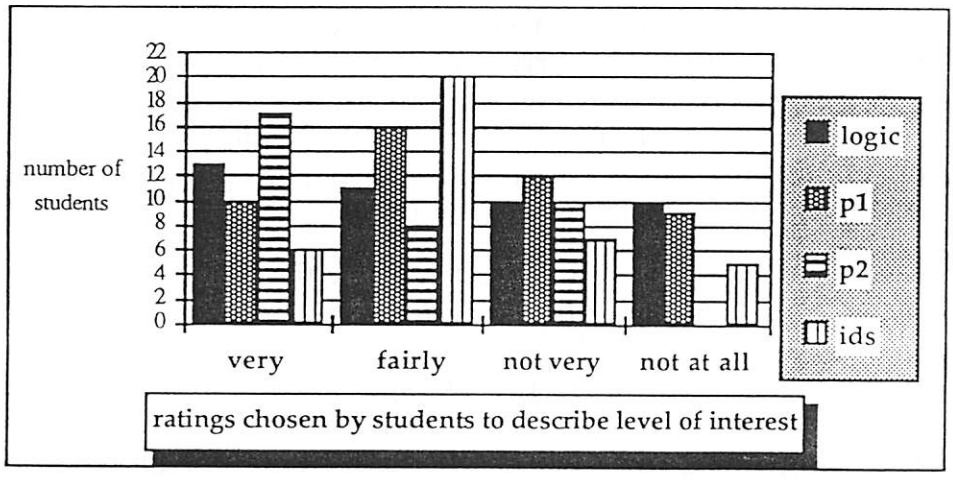

Figure 3 The interest factor

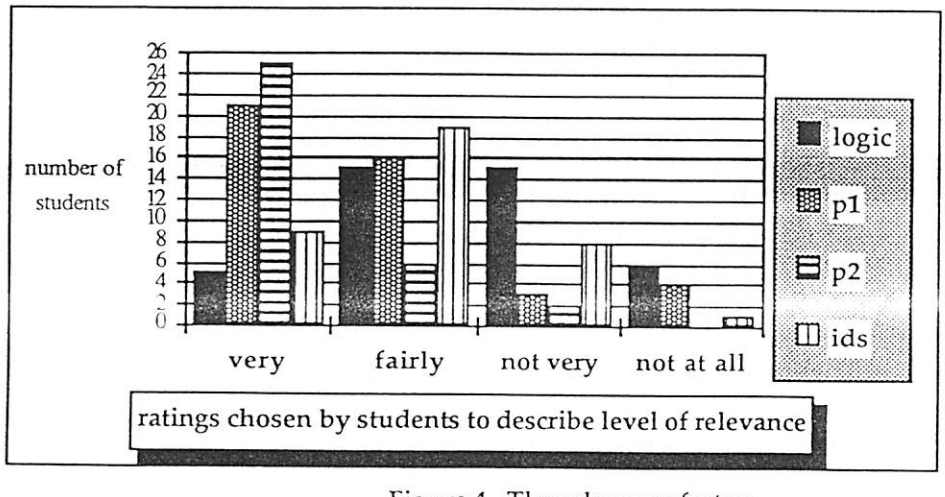

Figure 4 The relevance factor

Viewed at this level, it is relatively easy to pull out which courses were rated more interesting than others, which seemed to present most difficulty to most of those students and which these students considered most relevant. This however raises the 'why' questions - why did the interest rating for IDS peak above that of the other courses? Why did the relevance rating behave similarly for the course P2? Why did so many students find P1 not very or not at all easy, yet also rate that course to be fairly or very relevant? We attempted to gain insight on these aspects from the more detailed questionnaire comments and through the in-depth interviews in which we hoped to learn what criteria students used in rating the courses, what factors they took into account in rating a course as interesting, which aspects of a course meant that students found it 'very easy' or 'not all easy'.

Interpreting the data

In the event, not unexpectedly, there were no straightforward answers to many of the questions, but across the number of students interviewed, it was possible to discern certain recurring themes. These concerned factors which the students interviewed saw as contributory to their ability to succeed on the individual courses and indicated how those students perceived the relevance of the courses to their computer science studies. In summarising these themes it is convenient to classify them broadly as occurring at a general level, a subject level and, in relation to the perceived relevance of courses, at a personal level. This classification is to a certain extent an arbitrary one, since the relationship between the factors which affect students' learning, or their perception of their learning, is not one which can in practice be so easily unravelled.

At a general level, as we have chosen to call it, students saw the transition from school to university as an important factor in their progress. For many of them this meant adapting to a very different work style. Some saw this as a challenge. For others it posed problems. For them, the change from the relatively regimented framework of studies at school, to an environment where the responsibility for organising one's own work schedule, attending lectures, meeting deadlines etc., ultimately lay with oneself, had not been easy. At this level, the lecturing skills of those taking first year courses were most keenly appreciated. Interest in a lecture/course seemed to be stimulated by a clear presentation, the use of relevant examples where appropriate, the ability of a lecturer to 'communicate' with the students (as some interviewees phrased it), and the constant setting of each lecture in the wider context of the semester's/year's studies. Interest generated in this way seemed to have helped a number of those students interviewed to persevere at points where they would have categorised the course as 'not very easy'.

At a subject level, in many cases, students perceived that the difficulties they had experienced on courses were due to their lack of 'background' knowledge. Computing or programming experience was not a pre-requisite for the computer science degree course. A number of students who had no previous computing experience felt disadvantaged however, in relation to those students with computing experience and saw this as having been a source of their difficulties. These students felt that the time which they had to spend

familiarising themselves with the computing environment left them considerably less time to come to grips with the formal reasoning techniques to which they were being introduced.

Another source of difficulty at this subject level, as perceived by the students interviewed, was that given the time constraints of the first semester, the system of formal notation used in program proofs was too complex for them to manipulate confidently. While they were able to follow the steps, albeit with some difficulty, when these were explained to them, they found that carrying them out for themselves was a slow process with a high possibility of error given their lack of familiarity with the notation.

At a personal level this perception of the difficulty of formal reasoning was for some students strengthened by their view, at this early stage of their degree course, of its lack of relevance to the course. This is not immediately apparent if we look back at the charts (fig.1 to 3) showing the ratings given to the course P1 on the questionnaires. What we see there is that a relatively high number of students found the course 'not very' or 'not at all' easy, as was confirmed by the information we gathered from interviews. However, the charts also show that the interest ratings for P1 were not particularly low and that the relevance ratings were in fact high.

Students' changing perceptions

This apparent anomaly was explained in the course of the interviews and that explanation supported by additional comments on some of the questionnaires. A remark, often repeated, common to most of the interviewees, as well as appearing in some 'additional comments' slot on the questionnaires, was that the relevancy of P1 had become apparent only in subsequent semesters. It would seem that many of those students rating the relevancy of P1 as high on their questionnaires, were doing so with hindsight. From the interviewees, it became apparent that the formal reasoning approach to programming, introduced in P1 in their first semester, had not only seemed difficult, but that at the time, it had been hard to appreciate the relevance of the concepts being taught.

In conversation it became clear that for the majority of those interviewees with pre-university experience of programming, their view of the activity had been shaped by that experience. Presented with an approach to programming which advocated prior reasoning about the correctness of a program in relation to its specification, they had felt frustrated. Time not spent sitting at the keyboard actually working out a program by trial and error had seemed to them, at that stage, time wasted. For these students, this latter approach was what they expected to find on a programming course. Their initial reluctance to employ formal reasoning techniques was perhaps expressed most clearly in the comment 'I don't believe programmers in the real world do'. For them, coding in immediately their first pass at what they think would be a solution to the programming problem and then adjusting this program until it produced a satisfactory answer, was what 'real' programming was about.

This did not necessarily mean that for those without programming experience it was easier to see the relevance of using formal techniques. According to one such student, 'it was difficult to grasp abstract and theoretical aspects without having any of the basic computing concepts'.

Nevertheless, with or without prior programming experience, students' perceptions of the relevance of learning formal reasoning techniques were modified in the course of their studies. Of twelve students interviewed, ten mentioned that they had not found it initially easy to appreciate the relevance of the techniques they were being taught, but explained that by a later stage, in most cases near the end of the first year, they were able to appreciate the value of formal techniques learnt in their first semester.

Some implications

This study was undertaken as a first pass at viewing the teaching of formal methods from the perspective of those students who had followed courses which incorporated this approach to programming. The interest of the study lay in gaining insight into the difficulties which the students themselves felt they had encountered in learning formal methods; in hearing from them whether or not they felt those initial courses had influenced their own conception of what 'programming' was; in learning from their experiences in which ways they could most effectively be helped to appreciate and to put into practice this particular approach to programming.

There is a number of areas in which the the study has been useful. At a most general level, information gained reinforces and justifies the need to approach first year undergraduate teaching with the enthusiasm and careful preparation which those students interviewed had so obviously appreciated. At a level more specifically related to the formal reasoning contents of the courses in question, the information obtained raises several questions. In what way is it possible to help computer science students appreciate the extent to which the acquisition of initial knowledge and techniques (which are possibly unfamiliar and unintuitive to them) will make subsequent studies and projects easier? How best to avoid the situation, where, as one student commented, 'you only realise too late, that the initial work is so very useful. But it's very difficult to realise the relevance of something for the next step, when you do not know what the next step is.'

At a practical level, the study has shown that there is a need for a means of automating some of the processes involved in formal reasoning methods. It has shown that students lack confidence in manipulating formal expressions. Developing and introducing on-line tools which make those manipulations more transparent and reduce the likelihood of students' mistakes in performing operations on those expressions is a principal goal of the current research project, currently being undertaken at Queen Mary College. Such tools should help to alleviate some of the difficulties students experience in their initial encounter with formal reasoning techniques. Using such on-line help tools may also encourage an earlier modification of their initial perceptions of programming in the real world.

References:

Bornat, R. (1987) Programming from first principles. (series e.C.A.R. Hoare) pub. Prentice-Hall International.

Gries, D. (1991) Calculation and discrimination: a more effective curriculum. pp. 50-55 Comm. of the ACM, March vol 34 no.3.

Jones, C. (1986) Systematic Software Development using VDM. (series ed. C.A.R. Hoare) pub. Prentice-Hall International

Reeves, S. & Clarke, M. (1990) Logic for computer science. pub.Addison-Wesley.