Students' early attitudes and possible misconceptions about programming

David C. Moffat

School of Engineering and Computing,
Glasgow Caledonian University,
Glasgow, Scotland, UK
D.C.Moffat@gcal.ac.uk

Abstract. Programming can be unpopular with some university students of computing, who may then go on to graduate without good programming skills. This unpopularity threatens student recruitment into the core computing courses and professions, and may weaken the economy. It may be that negative attitudes harm the student's interest and confidence in programming, making for an unsatisfying learning experience.
In this pilot study, student attitudes towards programming, and possible changes in attitude, were investigated by means of a survey on a university's introductory programming course.
Results indicate that some students have negative attitudes toward programming, and programmers; and this applies to school pupils as well. A minority of the students questioned retained their frustration and dislike of programming throughout the course, but others came to love it in the end. Interpretation of the results leads to speculation regarding the quality of the teaching of programming, both at school and at university.

1 Introduction

In the UK and some other industrialised countries, student recruitment onto computing courses is falling. There may be various reasons for this, including demographic changes, curriculum changes at secondary school level, inadequate resources devoted to computing subjects, possible "dumbing down" and so on. But attitudes and possible misconceptions about programming might also have something to do with it.

When at university, following a computing degree, some students display an aversion to programming; when given the choice, they opt for project work that does not involve software development. Reasons given include being "uncomfortable with" programming, or not being able to program competently. This has been my own experience, in supervising student projects, or leading other kinds of student coursework in which programming is optional in some sense.

It may seem paradoxical, to suggest that some computing students dislike programming, because then it would be strange for them to choose to study computing at university in the first place. However, we should remember that students do not know the subject very well, and initially they do not share the more educated views of their teachers. Moreover, there are differing views amongst the teachers themselves. (At my institution, for example, I fall in the "hard-core" camp, because I consider that programming is the heart of computing; but there are colleagues who disagree with that.) Finally, the situation at schools (in the UK at least) is still worse, for there it is quite common for teachers to confuse computing with information technology. Being unclear about the difference, it would be no wonder that their students are confused, too.

Anecdotally, from talking with colleagues in staff room conversations, whether they see programming as essential to computing or not, it is generally accepted that some students show a distinct aversion to programming. However, there is little literature concerned with this issue. The present study is an initial attempt to explore student attitudes toward programming, and whether they have an impact on their learning.

While there is much attention devoted to the cognitive aspects of learning how to program, the affective aspects are relatively unexplored. Bennedsen and Caspersen (2008) sought correlations between some affective variables, including perfectionism, self-esteem and optimism, and performance in computer science courses. They found some interesting results, such as that self-esteem was correlated with success, as measured by good course grades. However, in present study is directed at a different set of affective variables, or attitudes: namely, self-efficacy, and social stigmas.

1.1 Confidence and self-efficacy in learning

It is known that *self-efficacy* (Bandura, 1994) is important in learning. The concept of self-efficacy relates to how people perceive their own capacity to achieve their goals, or goals that have been set for them. In an educational context, for instance, students have to learn things that may be quite difficult, and have to undergo examination to see whether they have learned successfully, and to what standard. While students are learning, they do not know whether they will succeed, in general, both because they are not yet familiar with the material to be learned, and because it is set at a fairly difficult level which should stretch the average student.

Self-efficacy is important in determining whether a person will succeed at a difficult task, because the strength of the belief can encourage the person to persevere, and so be more likely to overcome obstacles that appear along the way. It is related to confidence, but Bandura did not like to use the word "confidence" in his writings, because it can be used in a vague and general way, which is why he introduced "self-efficacy" as a new technical term. However, in ordinary language the concept may be rendered as a person's self confidence in being able to achieve a certain level of performance at some specified task, including both the strength or certainty of that belief, and the degree to which the level of attainment might be exceeded. In the context of learning how to program in a university course, for instance, a student may strongly believe that he will pass the course, but with only a low mark; or may initially believe that he could pass with a high mark, but believe that only weakly, so that he could be easily discouraged by an early setback.

The strength and degree of self-efficacy that a student has can clearly have a large impact on learning, therefore. There is also an interaction with affect or emotion, as minor successes and failures may confirm or defeat expectations. The emotion in turn can change self-efficacy as perceived by the student, and so regulate further learning performance. Bandura sees emotion as an important aspect of self-efficacy.

Programming is a challenging new skill to learn, for novices who have no experience of doing anything similar as far as they know, and so they can be extremely uncertain in their self-efficacy. Therefore one aim of this study is to question the nature of students' self-efficacy as regards learning how to program.

1.2 Negative attitudes about programmers may play a part

There are other reasons to be interested in the affective attitudes of students toward programming, such as any stereotypical views of programming and programmers. If students at a younger age, when still pupils at school, adopt society's prevailing attitudes toward programming, some of which may be seen as negative, then clearly there could be a large impact on study and career choices. Furthermore, if any students at university still retain such unfortunate attitudes when they have joined a computing course, then their learning could be retarded as a consequence, unless and until their attitudes can be corrected by experience. Any social stigma that may attach to a profession could undermine student motivation to choose that career, and study that subject.

1.3 Related research

Although there is not much literature about the affective attitudes of students toward programming, an interesting and related line of research is in the differences between boys and girls in scientific and technical subjects, and specifically in computing.

Beckwith and Burnett (2004) surveyed the literature on gender and learning, and drew several inferences about how best to design software development systems to benefit the strengths of both sexes. Beckwith et al. (2006) investigated a sex difference in tendency to "tinker" with, or playfully explore the features available in software products. Males tend to tinker more, which may account for their sometimes faster progress in learning to program; whereas females seem to be more reluctant to tinker with features they don't know about. This could be seen as a difference in learning style; or it might be interpreted as a form of risk-aversion in females, which would be a difference in affective attitudes of the sexes.

Differences between the sexes in self-efficacy in computing were found by Busch (1995), which may be attributable to the different personal histories, in that males had more experience of computing, and had benefited also from more encouragement from family and friends. In that case, gender stereotyping may be amplifying any difference in ability, via an affective route.

In a study of the personal histories of fifteen women who excel in their chosen careers of mathematics, science and technology, Zeldin and Pajares (2000) found that familiar role-models (male or female) and social encouragement were both factors in determining the career choices of the successful women. Each factor contributed to higher self-efficacy, which in turn led to greater resolution and higher performance.

The present study is not focused on women in computing, or on gender in any way; but the above literature is nevertheless relevant to our concerns, because it indicates that self-efficacy is a common thread in determining a person's success. Noting that apparent gender differences might not always be actually gender-specific, we can hypothesise that the above noted gender differences in self-efficacy and its causes may also be reflected within each of the sexes. Just as encouragement may help a girl to choose to study computing, and social stigma may put her off, the same factors may play a role with some boys more than others. For example, a boy who thinks that programming is difficult, and that programmers are unpopular or "nerdy," can be put off computing just as a girl might be.

This study is to investigate *all* students' attitudes to programming, therefore, and how they may change from school through to university. Do they have misconceptions, or experience misapprehensions about programming to the point that their morale and confidence suffer? By querying the students at key points during the semester, changes in their opinions may be tracked as they learn to program.

2 Method: to survey students about their current and previous attitudes

A questionnaire was given out to all students on an introductory programming module, which is to teach the object-oriented language C#. The questions are about experiences and opinions of programming at different times, as far as they can be recalled. In chronological order, these times were (a) from high school, (b) then just before the beginning of the module, (c) and then towards the end of the module (three quarters of the way through it).

2.1 The students

Glasgow Caledonian University is a "wide access" institution of higher education, which takes all sorts of students, including many from families that have never been to university. Consequently, they may be unusually prone to low self-efficacy because of the lack of role models in

their families. Students can also enter the university at second year directly, coming from local colleges of further education. Some of the latter students had good exposure to programming in languages like Java, and fair programming skills. They needed more practice, exposure to the C# language, and some deeper grounding in the principles of programming.

The introductory module that all the students were now following is a second-year module in object-oriented software development, in which the chosen language is C#. The students who started in the first year had a brief introduction to programming in that year, but it was slight in many cases, and some students can therefore be considered to be new to programming.

The module was taught in a way that would probably be considered unusual, compared with similar introductory modules (or courses) at other universities in the UK or elsewhere. There were some lectures, but the emphasis was on practical work, in which the students were expected to work in teams of four, and each team was to make a different, small, 2D video game. To do this they used a basic game engine written in C# and made publicly available by DigiPen,[1] which is a higher education institute for computer gaming, based near Seattle, USA. The DigiPen codebase uses DirectX technology, which was also new to the students, who were told the minimum they needed to know about it in order to be able to use the DigiPen libraries.

This all provided for a more authentic experience than is usual in student programming modules, and allowed them to take on a task that would show them what programming can really empower them to do. The coursework was therefore a relatively exciting assignment, giving the students wide scope for creativity and challenge, but also may have been daunting for some students. It will be useful to bear these details in mind when interpreting the results later on in the paper.

2.2 The questionnaire

The questionnaire was delivered on-line within the virtual learning environment that was being used to support the course. Questions were mainly of two formats. There were some Likert-style questions about skill or competence, and about attitudes and emotional reactions, like anxiety. There were also open-text questions, for less predictable responses. Some of them asked students to write in their own words what surprised them about programming, for example, or what their high and low experiences were.

The first few questions were about the student's background and previous programming experience. This was to see how many of the students were novices, and how many had significant programming skill.

Then, most questions were intended to detect changes in attitude from one time to the next, by asking students to recall their opinions about programming and programmers at earlier times in their lives. The three times of inquiry, and how they were expressed in the questions, were:

school "This question is about when you were starting your final year at school."
begin "This question is about your opinions in about August 2009, just before this module began."
now "This question is about now."

The questionnaire was answered by students about three-quarters of the way through the semester, after eight weeks of the intoductory module on (object-oriented) programming. This was the time referred to by "now" in the above.

[1] http://www.digipen.edu/gamers/tutorials/introduction-to-2d-video-game-development/ (accessed 07/May/2010)

3 Results

The survey was made optional to the students, and so not all of them took it. A total of 53 students started the module, and they were invited to answer the questionnaire on-line, in their own time. One student withdrew during the module, twelve did not attempt the survey at all, and two did open the survey online to look at it, but answered no questions. The response rate was therefore 38 out of 52, which is 73%.

Of those 38 students, eight answered only the first few questions, but then did not continue to finish the survey. Therefore only 30 of the 38 students answered the survey fully.

Because not all the students answered the survey, and some of them only answered some of the questions, the following results have been aggregated over all 38, in order to use all the data that was available.

3.1 Questions about programming competence

In order to find out about the background of the students, and their earlier self confidence, the following question was asked about their time at school, and the same question about the other two, later time points. It queries the student's *self-efficacy* regarding ability to perform a fairly straightforward programming task.

– This question is about when you were starting your final year at school. It is also about writing a small command-line, console program to implement a simple telephone directory, with name and number for each person. Back then, did you have the ability to write such a program?

Answers were given in the form of a Likert-scale, selected from: (s-ag) meaning "strongly agree"; (ag) meaning agree; (neut) for neutral, or neither agree nor disagree; (disag) for disagree; (s-disag) for strongly disagree; (na) for not applicable; and finally (un) for unanswered.

The time points (**t**) in the table are represented by **s** for school, **b** for beginning of the module, and **n** for "now" (which was towards the end of the module).

t	un	na	s-disag	disag	neut	agree	s-ag	median
s	0	2	6	10	6	12	2	neut
b	5	2	1	2	15	10	3	neut
n	8	0	0	0	2	20	8	agree

The figures in the table show how many Ss (students) answered the questions at each point on the Likert-scale: for example, six Ss strongly disagreed that they could write a small console program when they were at school. The total number of Ss who answered the survey was 38, but for each question some might not have answered, and so any missing values are shown in the **un** column. In the following calculations, the "not applicable" responses are also left out. For clarity, the resulting N values are shown below. The median point for each question is given in the last column.

From this table it appears that the students now are more confident than they were at school, or even at the beginning of the module, in their ability to write a small program. To confirm this a standard nonparametric test was run to compare each pair of rows. According to the Wilcoxon-Mann-Whitney test, the scores for **now** were (significantly) higher than for **beginning** ($W = 275$, $N_b = 31$, $N_n = 30$, $p = 0.001133$ two-tailed); and yet more significantly higher than for **school** ($W = 274$, $N_s = 36$, $N_n = 30$, $p = 0.000084$). The same trend is evident from **school** to **beginning**, but not significantly so ($W = 421$, $N_s = 36$, $N_n = 30$, $p = 0.07458$).

Most students have clearly increased their confidence in their ability to program — somewhat, from **school** to **beginning**; and a good deal more so, from **beginning** to **now**. We cannot say why their abilities improved in the first case; but the module has obviously been the reason for their most dramatic improvement (thankfully).

Note that the key phrase "confidence in their ability to program" deliberately obscures our assessment of their ability, by including the word "confidence". Because the students are answering the question themselves, and we are not independently assessing their programming skill at the earlier time-points, we cannot conclude anything directly about their skills, strictly speaking. This question is therefore interpreted as querying the students' *confidence* alone.

However, we may choose to infer that students have a fair idea of their own abilities in such cases, in which case we would indeed conclude that their increased confidence is justified by genuine learning. Then it is clear that, of those students that could not program before, most successfully learned how to, during the module.

One thing to note, on the other hand, is the small, hard core of students who still have low programming confidence (or skill) even towards the end of their first full programming module. There are two Ss who are still neutral on the question, even **now**.

We also note the wide range of abilities before the module began. At the beginning of the module, after their first year of higher education (or equivalent), already half the students were able to program. Some of them had learned quite recently, but most of them already knew some time beforehand, towards the end of their school education.

Therefore, quite a large spread of aptitudes is revealed, highlighting another problem that we have in teaching introductory programming at university. When there is a very wide range of abilities in a class, the teacher finds it very difficult to pitch the lessons at a good level – there can be no optimum level. This generally means that the weakest students will be daunted, while the strongest students could become bored and grow disdainful. The effect might be to put the students off programming even more.

3.2 Questions about general opinions of programmers

The following questions were about the students' former attitudes towards programmers and programming, to see if there was a perceived stigma attached to computing subjects, which might put students off choosing them.

First there were two Likert-scale questions regarding their opinions when they were at school, and now.

– Did you think of programmers (software developers of any kind) as cool?

t	un na	s-disag	disag	neut	agree	s-ag	median
s	5 2	1	2	15	10	3	neut
n	8 1	0	3	8	10	8	agree

It appears that there has been an improvement in attitude, with a shift in median from "neutral" to "agree", suggesting that more Ss now think that programmers are "cool".

This appearance was checked with the Wilcoxon-Mann-Whitney test, but not found to be significant at the 95% level ($W = 343$, $N_s = 31$, $N_n = 29$, $p = 0.09839$).

.

In order to let the students express their opinions in their own words, the following question was asked about each of the three time-points, to which responses could be typed in as free text.

– This question is about when you were starting your final year at school. What were your opinions about programmers and programming? (E.g. clever, boring, difficult, easy, nerdy, lucrative, important, misunderstood . . . ?) But in your own words or phrases, just off the top of your head.

There were too many answers to repeat, but here is a representative sample of what the students said, regarding each of the time-points. The responses were grouped into positive / neutral / negative categories, according to my judgement about the mood expressed in the attitudes. The selected responses reflect the sizes of the categories, and are shown in order from most positive to most negative.

– School [13 positive, 7 neutral, and 5 negative]
  1. Very easy and fun
  2. Intelligent, Difficult, Fun, Enjoyable, Creative
  3. Misunderstood, intelligent, problem-solvers
  4. Intelligent, i never thought i would be able to do the things they could
  5. "Taught what we needed to pass the exam, not what you need to program."
  6. I thought programming was boring, hard and the thing that someone else should do. Only the real nerds could understand it at all
– Beginning (before the module) [10 to positive, 13 no change or other, 1 negative]
  1. Programming is not as hard as i initaly thought
  2. not so difficult as my last year of school but i feel like i gotten rusty not programming as much i should have
  3. My orignal opinion that programming would be difficult to learn. I realised after some time that most programming languages are the same. It's just a case of learning the sytax. Just like having to learn grammer, when learning a new language.
  4. "The realisation of what a programmer actually does and how they have evolved"
  5. We only got visual basic and it was poorly taught as you didnt understand what it was really doing. So I was still anxious about starting programming. No change. {Note: this student seems to have been to college before entering the university directly into second year.}
  6. It gets far more complicated and complex than I thought, I no longer enjoyed it.
– Now (near end of module) [12 positive, 7 no change, 2 negative]
  1. The biggest change is that I am no longer afraid of it
  2. Programming isn't as difficult as i thought, it just requires a step back and some thought
  3. Using C# has shown me a different side of programming. I'm more used to the the web side of it - design, program and play, where as C# is design, program, compile, wait, re-build . . . I like it, but still prefer web development.
  4. My views have not changed
  5. No change.
  6. None really. Im still very confused with the whole thing

It was not possible to divide the responses into clear categories, because a matter like "difficulty" can be interpreted as positive or negative. When at school, the students had positive views of programmers in that they thought that they must be intelligent; but this could mean that some were discouraged from choosing the subject due to lack of confidence in their own ability.

Another issue is that the students did in fact choose to pursue computing at university, and so their views are not representative of those of typical students at school. It is notable however, that several students said that programmers were "misunderstood" by school pupils.

The question purposely prompted the students with some words that might be considered to be leading questions. For example, the words "boring" and "difficult" did recur in free-text answers as descriptions of programmers and programming. However, some other words did not (like "lucrative" and "important"). The word "clever" is in the question, and did occur in some students' answers; but so did the word "intelligent", which is a synonym, and a lot more often. There were also several new words that did not appear in the question in the form of synonyms or even antonyms, such as "fun", "enjoyable", "creative", and "problem-solvers". These observations taken together, while nothing like a formal analysis, nevertheless suggest that there was not a great overlap between the words in the the question and in the answers, and that the students were not being led to a great extent, but rather felt able to use their own words to express their views.

It is not shown which of the responses were from students who were familiar with programming or not, because the data is aggregated over the whole class. However, judging by their content, the more negative comments seem to be from pupils with some experience of learning how to program at school. There is a suggestion that they were not well taught.

Comments regarding the later time-points indicate that most students realised that programming was not so difficult to understand, but that the complexities can make it tedious, and slow to learn.

Coming up to date, the changes in opinion are mostly positive, with some that did not change, and sadly two that stayed negative, including the one student who remains entirely confused.

Some students have gained true insights about the nature of programming, suggesting that they might have previously suffered from significant misconceptions.

3.3 Questions about students' feelings toward programming

Turning to the students' own, personal attitudes toward programming, the following Likert-scale questions were about their possible anxieties at previous time-points, and about whether they like it now.

– Back then, would the thought of having to learn how to program make you anxious?

t	un	na	s-disag	disag	neut	agree	s-ag	median
s	7	1	1	9	4	12	4	agree
b	8	0	4	11	2	9	4	disag / neut

The median for **beginning** is exactly between the categories for "Neither agree nor disagree" and "Disagree", suggesting that students are not as anxious right at the beginning of the module than they were at school. This difference is not significant, however (Wilcoxon-Mann-Whitney, two-tailed: $W = 521$, $N_s = N_n = 30$, $p = 0.2728$).

Significant anxiety is clearly evident before the module began. Now that the students have learned much programming, the same question again would not be meaningful: however, they do appear to have changed their attitudes to the good:

– Do you like programming?

t	un	na	s-disag	disag	neut	agree	s-ag	median
n	8	0	2	1	4	8	15	s / agree

Nevertheless, there are still two who, whether they can now program or not, emphatically do not like or enjoy it. In fact, on checking the raw data, those two students are the same ones who, above, did not agree that they could write a simple console program.

3.4 Other questions

Some other questions included the students' experiences of learning programming, and more detail of their feelings about it, in their own words.

There is no space to detail the results from these questions here, commenting on all the answers individually. In summary, there was more talk of the difficulty of programming, but that on the other hand that it could be fun to make programs that work; also that it does not suit all personalities equally.

There was also a range of responses showing that some students came to find programming easy; and others find it still more difficult. Whether this is due to differences in personality or in aptitude is not clear.

4 Interpretation and further work

The purpose of this exploratory study was to see whether the programming students had negative attitudes towards programming, including whether it is difficult, or boring; and whether programmers themselves suffer from social stigma such as being boring, or nerds, or uncool. It was also a question whether the research method employed would be valid or useful, because asking people about their earlier attitudes are different times in the past, and comparing to their attitudes today, may be prone to confusion.

On the latter point first, we can conclude that the survey method has not failed, because trustworthy differences were indeed found between the various time-points. However, it may still be that there is some confusion about time-points, or memories may be faded, and if so then the results could only be weakened by that. Thus, we would not expect any false positives to result, but we should be aware that true differences may be harder to see, through the veil of memory. If anything, therefore, the results found by this method are an underestimate of the true effects that have occurred.

If the students on the introductory module are typical, then it will be a general problem that programming students have a wide variety of prior experiences. Contrary to its image of being difficult, programming is a skill that can be self-taught, and so there will always be a wide range of abilities in a programming module, which presents extra challenges to the teacher. In teaching to the stronger students, to keep them progressing, the weaker ones may be intimidated. Other subjects within computing do not suffer from this problem, which may help to account for why programming, in particular, is apparently difficult to teach.

Students certainly appear to have improved in their programming skills, both before they began their first serious programming module, and of course during it. We may infer this from their assessments of their own skills, which is a form of confidence in their abilities rather than direct evidence of them. The increase in (self-efficacy regarding) their skill was very significant, statistically speaking. To that extent at least, the module was a definite success; but there were also two students who still lack self-confidence. It turned out that they were the same two students who declared at the end of the module that they definitely do not like programming. At this stage it is too soon to be able to correlate their answers with their performance in the module, because those results are not ready yet. In future work, however, it would be valuable to examine the relations between affective attitudes and performance.

The affective attitudes that were probed in this study, by Likert-scale, included whether programmers were "cool" or not, and whether the prospect of learning how to program made the students anxious or not. A minority of students agreed that programmers were cool, with many taking no view; but there was also a significant minority that *disagreed* that they were cool. It appears that their views have changed to the good, so that now none of them strongly disagree, and more of them agree than did before, when at school. However, the difference was not statistically significant.

The questions about anxiety also appear to show an improvement between school and beginning the programming module at university, but this difference was not statistically significant either. The evident anxiety about learning to program is still a concern: it consumes the majority of the class, and it does not go away until the module is underway (if then). By the end of the module, however, nearly all the students seem to like programming, and most of them strongly agree with that statement. That is very encouraging, and may show that the students' fears were ungrounded: but this now begs the question as to where those fears sprang from.

In the free-text answers, students expressed their attitudes towards programmers and programming, showing a range of opinion from positive to negative. Again, it is a concern that computing students should harbour any such negative attitudes. However, it may be a normal state of affairs amongst students to be anxious about their learning goals. In further work it would be useful to scan the literature for research that could throw light on that issue, and then look to see if the situation is worse in computer programming than in other subjects.

Evidence from the free-text questions fleshed out the Likert-scale type questions a great deal. The range of responses from novice programmers showed some good depth of learning, and some impressive insights for young people with little experience. The affective content of their answers may have been cued by the affectively laden words in the question, in some cases, but in general the answers showed an independent turn of thought, and of word choices. The students can probably be trusted when they claim to find programming to be difficult but fun; or programmers to be misunderstood, or intelligent (or nerds). Some found programming boring, overly complex, or confusing. Even at the end of the module there were still some students with significantly negative attitudes, who dislike programming. They were only two out of the full 38, but one could say that is two too many.

One possible conclusion could be that the teaching on the module was just not good enough, and two or more students were not reached. On the other hand, it may be that there always are a few students on every course who do not like the subject, and programming might not be exceptional in this. Also, just because a student dislikes a subject does not mean that he is bad at it in general or that he would let it stop him from mastering the discipline. Further work could explore this issue, too. However, in this study at least, the two students who dislike programming at the end are the same ones who have weak self-confidence in their programming ability, and so we may hypothesise that negative affect correlates with lower ability. Whether this correlation holds up would be another matter for further work; along with which factor is a cause and which the effect.

Another possible conclusion could be that some students are simply bad programmers, and always will be, because they don't have the necessary mentality. For example, they might not be able to deal with abstractions well. Such a hypothesis is tempting for those teachers who may be reluctant to examine their own teaching, but there is no evidence in this study to support it.

.

In conclusion, this small study shows that negative affect is indeed a common and worrying factor in the psychology of student programmers.

The *effects* are not clear, but there may be impacts on performance when at university; and on career choice at or before university. A general stigma that programmers are "nerds" could discourage students from choosing computing subjects, but it appears that certain types of personality will tend to be attracted to programming even so. It may well be that some people have not only a preference but also a definite aptitude or talent for programming, and these are the ones who will really enjoy it.

However, even amongst students who have chosen computing for a career, it is deeply concerning that a proportion of them do not lose their fear of programming. They may seek refuge in other areas of computing, but will in that case always suffer a considerable disadvantage in their careers.

The *causes* of the negative affect that has been observed in this study are as mysterious as its effects, though one can speculate. Students may have acquired their attitudes from society at large, from Hollywood films, from friends and family, or even from their own computing teachers. Sometimes students complain of poor teaching at school, and in this study too there were a few comments suggesting that. It is credible, too, given that schools may easily confuse computing (where programming is a core skill) with information technology (where it is nothing of the kind).

However, there is no hard evidence in this study to suggest that the way school teachers teach how to program is any worse than, say, the way university lecturers (such as I am) do it. On the contrary, there was evidence in the class of this study that that a few students were not helped by their teacher (me) at all; in fact their state might have been made worse by him, if they have been turned away from programming as a result.

Whether at school or university, then, it is suggested by this study that poor teaching may be a significant factor in making pupils and students feel bad about programming. It is not an easy conclusion to accept, but it will be harder still to find out how it is failing. It is not likely to be the fault of programming teachers specifically, for they teach other subjects as well or better. Rather, we might blame the subject itself, and conclude that programming is simply difficult to teach. In that case, further research may help us to diagnose why that should be, and then we could do something about it at last.

5 Acknowledgements

Thank you to the students who answered questions about their attitudes, and to Thomas Green, John Richards, and the other participants at the PPIG-wip in Dundee for their comments. Warm thanks also to the reviewers of this paper, who took pains to make more suggestions than I could respond to in time for publication.

References

Bandura, A. (1994). Self-efficacy. In V. S. Ramachaudran (Ed.), *Encyclopedia of human behavior* (Vol. 4, p. 71-81). Academic Press.

Beckwith, L., & Burnett, M. (2004). Gender: An important factor in end-user programming environments? In *Ieee symp. visual languages and human-centric computing (vl/hcc'04)* (pp. 107–114).

Beckwith, L., Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., et al. (2006). Tinkering and gender in end-user programmers' debugging. In *CHI 2006.*

Bennedsen, J., & Caspersen, M. (2008). Optimists have more fun, but do they learn better? On the influence of emotional and social factors on learning introductory computer science. *Computer Science Education*, *18*(1), 1–19.

Busch, T. (1995). Gender differences in self-efficacy and attitudes toward computers. *J. Educational Computing Research*, *12*, 147–158.

Zeldin, A., & Pajares, F. (2000). Against the odds: Self-efficacy beliefs of women in mathematical, scientific, and technological careers. *American Educational Research Journal*, *37*(1), 215–246.