The Cryptic Crossword Puzzle as a Useful Analogue in Teaching Programming

Simon
School of Design, Communication, and Information Technology
University of Newcastle
PO Box 127, Ourimbah 2258, New South Wales
simon@newcastle.edu.au

Abstract

Contrary to the apparent beliefs of many students, computer programming and problem solving are not amenable to purely book learning. These skills can be acquired only by practice, and even then, students with an aptitude for programming will acquire the skills far more readily than those without. Unfortunately, aptitude is a concept that many students have difficulty appreciating. This paper describes a novel approach to helping students understand the concept.

Keywords: computer aptitude, cryptic crossword.

1 Does computer aptitude exist?

Many in the computing education community agree that there are students who take to programming and problem solving and students who don't. They further agree that these skills are so important that students who don't take to them tend not succeed in the course.

This phenomenon can therefore lead to high rates of dropout and failure, which have sparked much soul-searching and many papers. "Disturbingly high failure rates have been observed in first year Information Technology units at [Queensland University of Technology] over the past several years." (Brown 1996). A study at Monash University was "aimed at tackling perceived problems in the teaching and learning of first year programming. The main concerns were high failure rates . . ." (Carbone 2000). A paper from the University of Technology Sydney observed that "in many Australian universities, the failure rates for first year programming subjects are among the worst across the university. Many students become disenchanted, and either drop out or resort to plagiarism." (Lister 2000).

It must be acknowledged that this perception and conclusion are by no means universally accepted. One referee of the draft of this paper acknowledged "a bias towards the proposition that the core problem addressed in this paper is due to a faulty delivery on educators' part rather than a lack of aptitude in students" (Anon1 2003) while another wrote "I don't buy the premise" (Anon2 2003).

Nevertheless, the papers cited above, along with a wealth of others, show that many reputable educators share the perception that more students than expected have significant trouble with programming courses. Unless faulty delivery is endemic among teachers of programming, the existence of a relevant aptitude is appealing as a partial explanation.

This paper is written in the assumption that there is such a thing as programming aptitude. However, as one of the aforementioned referees pointed out, the paper's subject matter has potential pedagogical value even for academics who do not accept the existence of aptitude (Anon2).

2 The scope of this paper

Many papers have been written about ways of assessing the aptitude for programming and problem solving, from explicit tests (for example, Tukiainen 2002) to implicit indicators such as a certain level of attainment in mathematics (for example, Brown 1996).

But regardless of whether an assessment of aptitude is made, and regardless of the nature of the assessment if one is made, there remains the problem of persuading the less apt students that they will have great difficulty with the course; that much as they might like computers, they are unlikely ever to attain a comfortable grasp of programming, and might be well advised to realign their study in a direction that can derive more benefit from rigorous study.

This paper describes a simple analogue that can be shown to students to illustrate the nature of aptitude in a way that most of them seem to grasp in one brief lesson.

The paper does not describe a study or a research project. It simply describes, in the spirit of sharing useful ideas with one's colleagues, an analogue that has been used with some apparent success.

Through the paper, the word 'problem-solving' will often be used to avoid repetition of the more cumbersome 'problem-solving and programming'. Many good programmers would simply use the word 'programming' for this purpose, because they automatically problem-solve as they program.

However, part of the problem with students who lack the aptitude is that while they might eventually acquire the

Copyright © 2004, Australian Computer Society, Inc. This paper appeared at the *Sixth Australasian Computing Education Conference (ACE2004)*, Dunedin. Conferences in Research and Practice in Information Technology, Vol. 30. Raymond Lister and Alison Young, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

syntactic and semantic knowledge to become reasonable coders, they often fall short in the algorithm development that many good programmers do without thinking. The choice of 'problem-solving' is intended to emphasise this aspect of the problem-solving and programming aptitude.

3 Explaining the notion of aptitude to students

Many of my students, and I am not alone in this, have intriguingly little aptitude for problem-solving and programming. They like computers, and are more than capable in some areas of computer use such as, say, Web page creation; but when it comes to solving a relatively simple problem and implementing the solution in a programming language, they flounder.

If only these students recognised that they were floundering, and made a reasonably quick decision to shift their attention to an area in which they were more capable, all would be well. But most of them seem to feel that because they qualified to enter the course, they must therefore be capable of completing it. In one particularly troubling case, I saw a student devote nine years of his life to undergraduate study before finally being thrown out by the university for his continued inability to deal with the final-year courses.

In a first attempt to let students know that there is such a thing as an aptitude, and that not everyone who wants to program computers will be able to, I talk to them briefly about music and art.

With regard to musical or artistic talent, students readily accept that:

- many people don't have it;
- some of those who would like to have it don't, and end up either doing something else or scraping by in music or art;
- it has no correlation with academic ability;
- there is no shame in not having it.

However, students see music and art as somehow different from everyday activities, while they see IT as very much an everyday activity. They are conscious of the creativity required for music and art, while failing to see that problem-solving and programming might require creativity of their own. They feel that everyone who wants to do music or art has the aptitude, and that likewise everyone who wants to do IT will have the aptitude. Even among students who have studied IT for some time, the feeling persists that anything in IT can be learnt with enough study.

What was required, therefore, was a simple example of an everyday activity, not generally thought of as creative, that could be explained to students in a single lesson, and that would have a high chance of persuading them that aptitudes are not just for music and art.

4 The cryptic crossword puzzle

At the start of a tutorial, early in the course, I display an overhead of a cryptic crossword puzzle, such as the one in Figure 1. I then proceed to solve the puzzle, carefully explaining each answer to the class. By the time the puzzle is finished, many of the students are starting to look enlightened, to feel that they now know how to solve cryptic crossword puzzles.

Next I give the students a photocopy of a different puzzle. Some are quite surprised to find that they can't solve a single clue.

Now the parallel has to be drawn with problem-solving and programming; otherwise students will go away thinking they have been shown that they can't solve cryptic crossword puzzles.

Once it sinks in that being able to understand does not necessarily imply being able to do, I go on to draw some parallels between the cryptic crossword puzzle on one hand and problem-solving and programming on the other.

5 Lessons from the puzzle

5.1 Learning by seeing doesn't work

I started using cryptic crossword puzzles in this way many years ago, in another discipline, simply to illustrate the point that reading and understanding another person's solution to a problem does not in itself enable people to write their own solutions to similar problems.

Even if the techniques used are painstakingly explained, that is not sufficient to teach students to use the techniques for themselves.

Having already seen this illustrated, students have no trouble believing it. As the course proceeds, a gentle reminder now and then can start to persuade students who really do seem to be trying to learn to program just by reading other people's programs.

5.2 Learning requires much practice

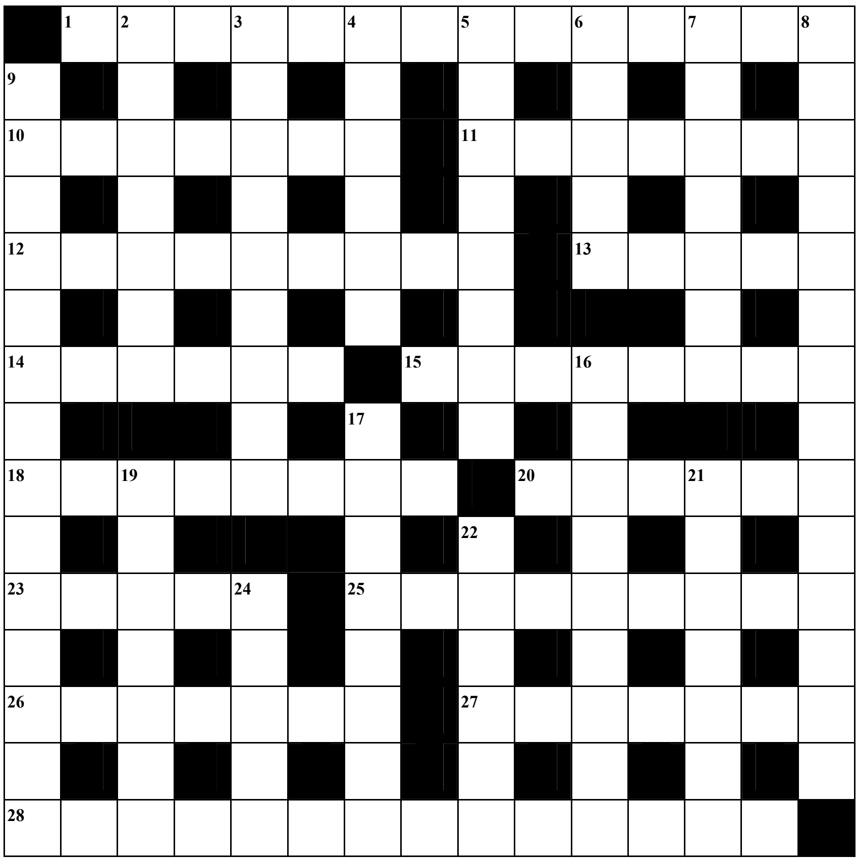
No matter how familiar one becomes with the techniques of cryptic crossword puzzle solving, one does not become a solver without a vast amount of practice. On the way, it can be very helpful to discuss the problems with a mentor or a fellow learner; but in the end, long-term dedicated practice is indispensable to acquisition of the skill.

Even with this practice, as some of the following points will illustrate, the skill will only be truly acquired by somebody who has the aptitude.

This is often the most difficult parallel for students to accept. Although they have just seen a counterexample, their limited recollections insist that virtually any academic subject matter can be learnt by reading enough about it. Yet the parallel with problem-solving and programming is a good one.

For example, the skill of debugging is seldom seriously addressed in textbooks. There has been work on systems that will help students understand compilation error messages (Kummerfeld 2003), but in the end, this understanding comes only with long exposure, which in turn implies a great deal of practice.

Ultimately, though, the benefit of this long exposure is not just in specific skills such as debugging but in the problem-solving skill as a whole.



Across

1 Dippy prophet (4,3,7)
10 Assess the size of step (7)
11 Art gallery's boss hangs "Dog on a hill" (7)
12 They fear coke's product when combusted! (9)
13 Wander about in mountains (5)
14 Where wet washing may be like some computers (2-4)
15 Tissue – painful to mangle it! (8)
18 Mistletoe is one – it makes Australian parties go with a swing! (8)
20 Very old Briton working here in NSW (6)
23 ABC, for example, make fightback (5)
25 Curate with his roll performing this (9)
26 Like the start of Ulysses! (1-6)
27 Extremely thankful to be employed in dull work? That's a laugh! (7)
28 Bananas Sardinians cook as snack (4,3,7)

Down

2 Where one may find butter experiencing continued success (2,1,4)
3 They're particularly small – but no matter! (9)
4 Has money invested in dogs (6)
5 Family groom's friends put on some leather (8)
6 Dad's about right – but he snapped during WWII (5)
7 Vivid, say, description of campers? (7)
8 It has many branches of repute – internet bank (10,4)
9 Engineer with pump or hammer – he's big but all flap – no takeoff! (7,7)
16 They arouse artist besotted with Goa (9)
17 Waited and listened (8)
19 Chatter about type of wheel (7)
21 Legendary hero trains furiously around end of August (7)
22 What'll sound like this tree? (6)
24 Reddish monkeys jump up around one (5)

Figure 1: Sunday Times Crossword No 225 (Australian 2001)

5.3 General knowledge is required

Few crossword puzzles, cryptic or otherwise, can be solved without a better than average general knowledge. In figure 1, for example, 1a (clue 1 across) requires a fairly common piece of knowledge about the New Testament; 20a requires some knowledge about the ancient peoples of Britain, along with some geographical knowledge of NSW in Australia; 4d requires some knowledge of currencies; 6d requires knowledge of a famous Australian of 60 years ago; and 21d requires knowledge of an Arthurian legend.

Much to the surprise of many programming students, problem-solving also often requires a reasonable general knowledge. A classic example is the algorithm for determining whether a year is a leap year – which the programmers of the world's most widely used spreadsheet software haven't managed to get right yet.

5.4 Specialist knowledge is required

The obvious specialist knowledge required for solving crossword puzzles is vocabulary. Solvers need to be familiar with a great many words and their sometimes multiple meanings.

In addition, some crossword puzzles have a theme, such as the fauna of New Zealand or the works of Lewis Carroll. Solving these puzzles is greatly facilitated by specialist knowledge in the theme area.

The obvious specialist knowledge for programming is the syntax and features of one or more programming languages. Students generally accept this – indeed, they often seem to believe that this is all the knowledge they require in order to be able to program well.

But most computer applications have a theme, too. They are written to address needs in areas in which the programmers are generally not experts; to write them properly, the programmers need to become quite familiar with those areas.

Interestingly, at the educational level students sometimes respond to this need more dramatically than they should. A number of my assignments in different subject areas have specified that a company keeps a number from 1 to 10 to register the degree of sycophancy required for each of its clients. Many students choose to ignore this piece of information because they don't understand it, when in fact all they need do is create an integer variable or attribute called sycophancy.

5.5 Standard techniques can be applied

It becomes clear to students after a good look at a single cryptic crossword puzzle that certain standard techniques crop up with monotonous regularity.

In figure 1, for example, nine of the 28 clues are anagrams. However, no clue says explicitly 'make an anagram of these words'. Instead, they use signals, some fairly obvious (cook, go with a swing) and some more obscure (hammer, roll, and even bank), that will eventually tell the assiduous solver to mix up the letters of certain other words in the clue.

Even then, the answer need not be obvious. A nine-letter anagram has as many as 362 880 possible arrangements, from which the answer might take quite some time to emerge.

The analogy with problem-solving and programming holds good. There are many classic algorithms and standard techniques in problem-solving and in programming; it is not always obvious when one can be fruitfully applied; and even when that much becomes clear, the programmer has to work out in which particular way to apply which particular technique.

5.6 Many resources can be used

The best crossword puzzlers need no resources other than their minds. The rest of us soon accumulate a bewildering array of resources.

After a good dictionary or two, the next acquisition might be a reverse dictionary or a crossword dictionary, both of which can lead from the definition to the word rather than the converse.

Then there are anagram dictionaries, which list the jumbles of letters alphabetically, following each with the word or phrase that it anagrams to.

Books of lists can be handy: lists of authors, of politicians, of artists; lists of countries, towns, rivers, lakes, mountains; lists of birds, of animals; lists of mythical folk such as the muses, the seven dwarfs, the gods of Olympus.

A good atlas is a must, and a postcode listing can be even more comprehensive.

This is a hard lesson for many programmers, who like to think that once they have learnt a given programming language for a semester or two they can use it to do anything that might be asked of them.

The basic knowledge of programming skills is seldom enough for serious applications, and programmers will often have to resort to sundry resources to supplement their own knowledge and skills.

5.7 Answers require testing

It isn't immediately obvious that crossword answers need testing, but in fact they are subject to testing of the most stringent nature. The prime test of each answer is whether it interlocks correctly with the other words on the puzzle.

At times this test can be used to help solve other clues: 'this looks as though it might be vivisection; if it is, that other word will start with a v'. A hypothesis of this sort can lead to a solution for the other word, which is then a fairly strong indication that the first word was right.

Every programmer knows that programs need testing, but not all programmers test as stringently as possible, and, in particular, programmers can easily overlook the testing of interfaces between modules. The crossword puzzle can be a useful reminder of the need to do this.

5.8 Insight is required

Solving cryptic crosswords requires a particular kind of insight, an often perverse way of thinking about the problem.

This is one of the features of problem-solving and programming, although many good problem-solvers would prefer to label their thinking as lateral rather than perverse.

5.9 An analytical capability is essential

Some people solve problems intuitively, others analytically. While intuition can be extremely helpful in solving crossword puzzles, it cannot take the place of clear analytical thought.

In the same way, intuition can be a useful addition to the arsenal of a problem-solver, but that arsenal will be fairly ineffective if it is not founded on solid analytical skills.

Furthermore, analytical skills are reasonably independent of the problem domain. This point scares some budding programmers: they can see instantly that they might never acquire the analytical skills for solving cryptic crossword puzzles, and they consequently become a little more receptive to the possibility that they might never acquire those same analytical skills for problem-solving.

6 Outcomes from the lesson

This lesson, generally delivered some time in the first few weeks of a first programming course, has had several different favourable outcomes.

First, some students who later decide to give up on IT refer back explicitly to that lesson. They do this without rancour, accepting that they do not appear to have the aptitude for problem-solving, and that they should find an area to which they are better suited. While some academics are loath to lose any students, I am actually glad to see such students leave, because I do believe that they will be better off in a different area.

Fortunately, it is not only students without the aptitude who can benefit from the lesson. Every student who sees the parallel accepts in one simple lesson that learning programming involves far more than simply learning the syntax and semantics of a specified programming language. This is a lesson whose value can last for the remainder of the course and beyond.

Finally, just now and then a student thanks me for introducing her/him to the challenge and the pleasure of cryptic crossword puzzles. This tends to happen some years after the lesson, and is always gratifying.

7 Is the analogue really applicable?

The analogy of the cryptic crossword puzzle seems to help a large number of students to grasp the notion of aptitude, and to accept that aptitudes aren't just for such overtly creative callings as music and art.

This can then help them to accept that there is nothing to be ashamed of in lacking the aptitude for problem-solving and programming; it does not mean that a student is not bright, is not smart; it just means that the student lacks that particular perverse way of thinking that some people label 'computer aptitude'.

When I have simply told students that some of them just won't have what it takes to be programmers, those who eventually drop out seem to feel somehow inferior.

On the other hand, when I have illustrated the message with this analogy, they seem more ready to accept that problem-solving and programming are simply not aligned with the way they think, and they generally seem much happier to go on to some other area of interest.

It would seem, therefore, that the analogue is applicable, at least inasmuch as it has some positive outcomes.

8 Conclusion

I am not the first to have seen the analogy between solving puzzles and solving computing problems.

At one end of the scale, the University of Malta, in advertising its B.Ed. (Hons) in Computing, says "If you do not enjoy solving logical puzzles then this is not the option for you."

At the other end, and more than 60 years in the past, Bletchley Park, the group led by Alan Turing to crack the Enigma cipher, recruited some of its codebreakers with a speed test on a cryptic crossword puzzle from the Daily Telegraph (Singh 1999).

So while I first saw the cryptic crossword puzzle as pure analogy, it might in fact be something more; perhaps it could one day be harnessed as one of a set of predictors of computer aptitude.

9 References

Anon1 (2003): Blind review of this paper.

Anon2 (2003): Blind review of this paper.

Australian (2001): Sunday Times Crossword No 262. *The Weekend Australian*, July 2001. [Several attempts to obtain permission to reprint this puzzle have gone unanswered. The author interprets this lack of response as tacit assent.]

Brown, C., Hogan, J., and Hynd, J. (1996): Prediction of Student Performance Using Neurocomputing Techniques. *Proc. First Australasian Conference on Computer Science Education*, Sydney, Australia, 273-279, ACM Press.

Carbone, A., Hurst, J., Mitchell, I., and Gunstone, D. (2000): Principles for Designing Programming Exercises to Minimise Poor Learning Behaviours in Students. *Proc. Fourth Australasian Computing Education Conference*, Melbourne, Australia, 26-33, ACM Press.

Kummerfeld, S.K., and Kay, J. (2003): *Proc. Fourth Australasian Computing Education Conference*, Melbourne, Australia, 105-111, ACM Press.

Lister, R. (2000): On Blooming First Year Programming and its Blooming Assessment. *Proc. Fourth Australasian Computing Education Conference*, Melbourne, Australia, 158-162, ACM Press.

Malta (2003): Web page for The University of Malta B.Ed. (Hons) in Computing. http://www.educ.um.mt/computing/index.asp Accessed 5 September 2003.

Mazlack, L.J. (1980): Identifying Potential to Acquire Programming Skill. *CommACM* **23**(1).

Singh, S. (1999): *The Code Book*. Fourth Estate, London, England.

Tukiainen, M., and Mönkkönen, E. (2002): Programming Aptitude Testing as a Prediction of Learning to Program. *Proc. 14th Workshop of the Psychology of Programming Interest Group*, Brunel University, 45-57.