Searching for Tomorrow's Programmers

Michael de Raadt
University of Southern Queensland, Toowoomba, Australia

deraadt@usq.edu.au

Abstract

Interest in beginning a career in programming is declining. Programming is no longer selling itself to novices at high school and university levels as well as it once did. This paper considers why this is the case, what may happen if this trend continues, and what can be done to restore interest for the next generation of programmers.

Keywords: novice programming, teaching programming languages

Introduction

I thought to myself "Why am I even here?" It was 2am and I was on a minibus driving through the outskirts of Colombo, Sri Lanka. The driver hurried along at a frenzied pace taking every opportunity to overtake. The minibus had worn shock absorbers and the road was poor, so every bump and pothole lasted longer. The speed limit was marked as 32km/h but the traffic seemed to be travelling at 70-80km/h. At one stage we slowed and I hoped that we had arrived at our destination, but it was only a police speed checkpoint. The driver turned towards us and, in his limited English, chuckled "High speed" and soon the rough ride continued.

With me were three of the finest programming geniuses the Australian high school system had to offer. The boys had been selected from the top performing competitors in the Australian Computer Programming Competition (ACPC), a national programming competition for high school students. The competition is now in its twentieth year. In its history the competition has adapted to changes in languages, operating systems and methods of communication. From a paper based competition where questions and solutions had been sent through standard mail, the competition has used fax, email, and now runs online with automated question distribution and marking. Interest in the ACPC has been diminishing for a number of years. At its peak during the early to mid 1990s the competition enjoyed participation of over 4000 students from around the nation. In 2003 interest in the competition had lessened. After several advertisements in teachers' journals and postings to computing teachers' mailing lists, together with prize incentives including cash and computers, the participation was just over 300. As organisers we have considered discontinuing the competition due to low levels of interest.

A team of three had been selected from ACPC competitors to compete in an international programming competition: The International Schools Software Competition (ISSC) held in Sri Lanka in December 2003. The competition runs in conjunction with the South East Asian Computer Confederation (SEARCC) Conference. SEARCC is a parent body to national computer societies including the Australian Computer Society (ACS). The ACS was supporting the Australian team as it travelled to compete. As

one of the ACPC organisers I had been given the role of Team Manager for this Australian team.

The ISSC has also been affected by world affairs. In 2002 the competition (together with the SEARCC Conference) was cancelled due to SARS fears in Asia. In 2003 the competition went ahead, but the SEARCC Conference was again cancelled. Only four teams competed in the ISSC; two from Sri Lanka (the host country), one from Thailand and, of course, one from Australia.

Introductory programming courses can be seen as starting point to a Computer Science degree and a future career in software development. Introductory programming courses at universities around Australia are also witnessing a decrease in interest. (This paper refers to courses as a single semester long period of study. This may be equivalent to a unit, subject or paper in other institutions.) In 2001 and 2003 a census of introductory programming courses around Australia was conducted (de Raadt, Watson & Toleman, 2004). As well as measuring popularity of languages taught, use of tools and teaching methods, the census also measured student enrolments in introductory programming courses. Courses which continued between 2001 and 2003 experienced an average reduction in student enrolments of 28%. When the census was repeated in 2003 instructors were not reminded of their previously reported enrolment figures, nor were they told that there was a general trend of lessening enrolments developing, yet all but three participating Australian instructors reported a reduction in student numbers in their courses. Some courses experienced a reduction as high as 71%.

This paper is divided into sections which attempt to convey a rational chain of reasoning. The following section examines why less students are interested in programming. This is followed by a section which considers consequences academia and industry face if this trend continues. The penultimate section suggests potential strategies that may be implemented to reinvigorate interest in programming among students. Finally conclusions are made.

Why?

Whether the reductions in involvement in the ACPC and enrolments in introductory programming courses indicate an ongoing trend or form part of a cycle, it is worthwhile considering factors influencing senior high school students, high school leavers, and the general population contemplating a career in programming. Why are people, particularly high school students, less interested in programming?

One reason may be that computers are now more common-place in homes, certainly more so than when the ACPC saw its peak in interest in the early 1990s. Many students who are completing high school today have had a computer in their home since their birth. For years, computer programming gave a first taste of computers to those eager to use a computer. In order to create a computer game a novice may have attempted entering code from a codebook or creating new programs from scratch. When most students did not own their own computers, computers in schools and universities, and courses which involved programming were popular. Students no longer need to be at school or write their own programs in order to experience what computers can do. Computers are now powerful multimedia instruments and the games and software packages that users can buy off-the-shelf, or even download for free from the World Wide Web, are far more complex than what a single novice programmer can create. For the "now generation" or "generation Y" putting effort into learning how to create usable software seems fruitless. Users of computers now fall into a broad spectrum covering many levels of ability ranging from expert programmers through to occasional email checkers. Future programmers must now traverse a vast pool of computer users and paddle their way to the deep end.

Within universities, the audience for introductory programming courses has varied for several years. As computing became a more popular practice, and computing skills became highly valued, students from all fields were encouraged to develop some amount of experience with com-

puters. During the 1980s and into the 1990s the best way of providing these skills was through training in programming. A notion developed that all computer users could benefit from some amount of programming literacy. Because of this, programming was referred to as “The New Latin” (Sleeman, 1986). Novices who would be trained to become expert programmers were joined by a new group of students who were exposed to not more than a semester or two of programming instruction. Today these same never-to-be-expert programmers are no longer expected to have any programming experience. Instead, these students are catered for by end-user applications courses which instil word processing, spreadsheet, Internet client use and electronic presentation skills. While this approach covers the needs of most future end-users, it reduces the number of people exposed to programming and potentially attracted to the profession. It also denies many people the understanding of the power of programming.

Computers have great potential as learning tools. Computers in homes and schools are loaded with applications for younger users but according to Kafai and Soloway (1994) most educational packages are far from complete educational toolkits. While computers are recognised as essential business tools, for many younger people they are no more than powerful gaming machines.

Many schools are now establishing computer labs – rooms full of many and various personal computers. Yet schools rarely consider the costs of maintaining computers after their purchase and teaching staff rarely have the time or expertise to keep such a number of disparate computers working in a consistent manner, so "by the end of the month, these machines become misconfigured, raggedy piles of plastic," (Soloway & Norris, 1999). Schools are the first place where most students are formally taught about computers. We must question whether they are receiving a positive exposure to computers. Are schools nurturing the skills of tomorrow's programmers? Is there room in primary and high school curricula for teaching programming skills? Are modern programming languages and tools suitable for teaching logic, syntax and problem solving skills to younger programmers?

The Oliver Internet Job Index (Oliver Recruitment Group, 2003) reports that jobs advertisements in the Information Technology and Telecommunications sector peaked in June to October 2000, but fell to less than a quarter of these levels in December 2001 and have remained low since then. More specifically within the Software Development and Engineering sub-sector, job advertisements in November 2003 were equivalent to 17.5% of the number in May 2001. According to The Grad Files Report (Graduate Careers Council, 2004) 86.6% of Computer Science students who graduated in 1998 were in full-time employment in the following year. Only 68.1% of 2002 Computer Science graduates were full-time employed in 2003. This trend is illustrated in Table 1. While students entering tertiary study will probably not examine figures this closely, it is clear to the general public that the gloss and glamour of a career in programming is not what it once was. Students want to be assured that the result of study at university is a well paid, respected position.

Table 1: Employment of Computer Science Graduates from Grad Files Reports 1999-2003

Year graduated	Year measured	In full-time employment	Seeking full-time employment, not working	Seeking full-time employment, working part time or casual
2002	2003	68.1%	16.4%	15.6%
2001	2002	70.5%	16.4%	13.1%
2000	2001	81.0%	12.4%	6.7%
1999	2000	88.2%	7.6%	4.2%
1998	1999	86.6%	8.2%	5.3%

The Australian media is quick to point out when Australian jobs are under threat. Even though only a minor proportion of Australian IT jobs are currently under threat, this is a key issue is the popular press. In April 2003 Australian telecommunications company Telstra claimed that no Australian jobs would move offshore. On April 22, Telstra's chief information officer Jeff Smith said "Telstra has no intention of replacing any of its IT employees with staff of external IT providers, from India or elsewhere in the world." Yet in September of the same year Telstra broke off ties with its IT partner company IBM and shifted 180 jobs to Indian IT group Infosys in an effort to reduce its IT costs (Sainsbury, 2003).

For many of the general public, suffering through a computer crash and the loss of work that accompanies this event is an accepted consequence of relying on computing technologies. Millions of point-and-click computer users curse the software they use. As computer applications and operating systems become bloated to sell a newer version and the number of people worldwide using those applications grows, erroneous segments of code inevitably rise to the surface. This does not reflect well on the people who created the software and taints the image of the programming profession. The successes and failures of software companies are now public knowledge. When the "Millennium Bug" was suggested as a source of catastrophic danger to software systems and the infrastructure that relied on these systems, many in the general public panicked. While most non-programmers had no understanding of the nature of, for example, short dates in legacy COBOL code, the consequences of poorly conceived code were publicly acclaimed. Even though little damage was felt when computer clocks turned over to the year 2000, the mud still sticks. A more real example of failure has been the overcapitalisation of 'dot com start-up' companies. While these companies innovated ideas and technologies, they were not strong enough to survive economic downturns and withdrawal of support from the general community that this brought.

We can no longer rely on a mystical aura to draw novices to an 'unknown world' of programming when that world is now on display to all.

What If?

Involvement in initiatives such as the Australian Computer Programming Competition and enrolments in introductory programming courses can be seen as an indicator for the future of programming instruction within schools, universities and the programming profession.

If a trend of decreasing student enrolments in introductory programming courses were to continue, then this would alter the nature of computer science education. If there is less interest from students, then this will diminish the justification for research into programming and programming pedagogy. If teaching of programming decreased in high schools and universities then instructors would lose a major instrument for problem solving instruction. With less people interested in programming now, there will be less future programming educators within high schools and universities. These factors could compound the problem further.

Less student interest in programming now will have a flow-on effect to industry in coming years. The major consequence will be programmers becoming rarer in Australia. Many positions with the Information Technology industry do not require employees to write programs, but when designing, managing and selling software systems, an understanding of the underlying construction of such systems is reasonably necessary. While having fewer programmers may provide higher wages for local programmers, many programming jobs will go overseas where wages are lower and interest in programming is higher. Quality of software may suffer with programmers being hired with less potential and poorer qualifications. Corporations may be forced toward a greater reliance on off-the-shelf products rather than custom solutions. These factors will lead to a poor reflection of the profession and again compound the problem.

What Can Be Done?

The best solution to these problems is to raise interest in programming now. Greater involvement in programming in schools at primary and high school levels is needed to promote programming as a challenging and respectable profession. Schools and supporting agencies should encourage involvement in new initiatives similar to the ACPC, for ordinary students through to geniuses. A list of competitions and programs is shown in Table 2. Often high school students who seek a programming challenge are forced to look to online collegiate and professional competitions. Such competitions run perpetually and allow programmers to create a profile of successfully solved problems. An initiative to create such a "programming arena" with realistic challenges for high-school students, supported by government and industry, could be used by teachers and parents to challenge keen novice programmers.

Table 2: Programs Encouraging Interest in Programming

Program	Web
Australian Computer Programming Competition	http://www.usq.edu.au/acpc
UNSW/CSE High Schools Programming Competition	http://do.cse.unsw.edu.au/progcomp
Australian Informatics Olympiad	http://www.amt.canberra.edu.au/aicfact.html
Jade Kids	http://www.jadeworld.com/education
ACM International Collegiate Programming Contest	http://icpc.baylor.edu/icpc
TopCoder (Collegiate/Professional Programming Competition)	http://www.topcoder.com

Each state in Australia has an organisation of high school computer science teachers and there is also a national umbrella body. These organisations support their members by providing teaching resources and running conferences. Such bodies should be further supported by government and professional organisations so that their teacher members can achieve their complex, often overworked occupations. Computing teachers need to be supported in establishing and maintaining proper computing facilities for their schools so that students can get the most out of computer use.

Getting students to write actual programs is possible at all levels of schooling. Success with high school students has already been proven (Finkel, Hooker, Salvidio, Sullivan, & Thomas, 1994; Hussey, Leadbetter & Purchase, 1996). Getting students to write programs is necessary in order to illustrate the importance and potential of programming. Having teachers who can teach real programming skills is also necessary. Having realistic and reliable environments conducive to student learning of programming is also a major consideration.

When attempting to attract novices to programming, the content of programming courses in high schools and universities must also be considered. It is no longer sufficient to teach students using tasks that merely incorporate the constructs involved in a programming language. Programming needs to be portrayed as practicable and interesting. Students need to be able to create a self image that involves a potential to write programs, or at least understand the power of programming, rather than being simple end-users. Programming tasks should be interesting and relevant to students' lives, demonstrating the full potential of computers (Guzdial & Soloway, 2002; Parlante et al. 2003). Examples include using web technologies, games, robotics (eg., LEGO Minstorms), sound and/or graphics as part of practical tasks. This approach will also aid in student retention during and after a first exposure to programming. Programming tasks have to be isolated from other general computer tasks, such as word processing, so that students can understand the significance of programming and its influence on a computer.

At university levels it is a critical time to reconsider the audience for programming instruction. While non-computing majors may no longer need full training in programming, they can still benefit from some programming training, particularly as it aids training in problem solving. New courses which form a middle ground between applications courses and exclusive programming courses would be a valuable addition to students' study and at the same time encourage a greater student body to recognise the strengths of programming (Joyce 1998; Mayer, Dyck & Vilberg, 1986).

Computer Science Education and the study of the cognitive processes involved in the teaching of programming are still in a stage of infancy. Many students struggle with their learning of programming and many teachers lack a full potential to teach students how to program. Programming pedagogy must remain as a prominent and active area of research. Cognitive studies of expert and novice programmers have been conducted (Fix, Wiedenbeck & Scholtz 1993; Soloway 1986). An investigation of the cognitive abilities relating to programming among students at various ages would allow programming instruction to be focused and encouraging at the earliest stages of computer use. Strengthening research in how programming is taught will not only benefit students, but also support more people as they teach programming. If the importance of this field of research decreases then funding to support such research will diminish and channels to output such research will become scarce.

Ultimately the greatest draw card for people considering entering the profession of programming is a good reputation. The image of programmers needs to be portrayed as accurate (not the people who create the software that makes your computer crash), serving the interests of the public (not serving the needs of a giant corporation), and with a potential for reasonable earnings. Programming needs to be portrayed as challenging, enjoyable and achievable. This image needs to be marketed through professional societies and targeted at those considering entering the profession and also the general public. Industry bodies need to be encouraged to employ programmers with confidence. The benefits that programmers can bring to a corporation needs to be promoted and the importance of keeping programming jobs within Australia needs to be made clear. If demand for programmers increases, interest in becoming a programmer will soon follow.

Our Future

The competition in Sri Lanka was a success. The team from Australia placed a close second. The boys were enthusiastic about their performance and were all looking forward to further study and a career in programming. Although I was accompanying a group of young people far beyond many of their peers in skill and potential, I could still see within them the characteristics of a future generation of computer users and, more specifically, programmers. This new generation will no longer treat computers with awe, but instead use computers as tools to get a job done. It is this generation that will either allow computer programming to stagnate or take it to new and greater heights. So to answer my own question, "Why am even I here?", I am here to encourage the next generation of programmers and hopefully share my enthusiasm for programming.

Acknowledgements

I would like to thank the Australian Computer Society for their support, Patrick Coleman, Chris Ebneter and James McGill (2004 Australian ISSC Team), and Mark Toleman and Richard Watson for their assistance in writing this paper.

References

de Raadt, M., Watson, R. & Toleman, M. (2004). Introductory programming: What's happening today and will there be any students to teach tomorrow? To Be Published in *Proceedings of the Sixth Australasian Computing Education Conference (ACE2004)*, Dunedin, New Zealand, Australian Computer Society.

Finkel, D., Hooker, C., Salvidio, S., Sullivan, M. & Thomas, C. (1994). Teaching C++ to high school students. Selected papers of the *Twenty-Fifth Annual SIGCSE Symposium on Computer Science Education, Phoenix*, AZ. New York, NY, USA: ACM Press.

Fix, V., Wiedenbeck, S. & Scholtz, J. (1993). Mental representations of programs by novices and experts. *Proceedings of the Conference on Human Factors in Computing Systems*, Amsterdam, The Netherlands. Boston, MA, USA: Addison-Wesley Longman.

Graduate Careers Council (2004). The Grad Files Report 2003. Retrieved January 30, 2004, from http://www.gradlink.edu.au/content/download/1567/5692/file/GFiles2003.pdf

Guzdial, M. & Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM, 45*(4), 17-21.

Hussey, A., Leadbetter, D. & Purchase, H. (1996). Learning object-oriented programming in six hours: an experience with school students. *Proceedings of the Second AustralasianCconference on Computer Science Education*, Sydney Australia. New York, NY, USA: ACM Press.

Joyce, D. (1998). The computer as a problem solving tool: A unifying view for a non-majors course. *Proceedings of the Twenty-ninth SIGSE Technical Symposium on Computer Science Education*, Atlanta, Georgia, USA.

Kafai, Y. & Soloway, E. (1994). Computational gifts for the Barney generation. *Communications of the ACM, 37* (9), 19-22.

Mayer, R. E., Dyck, J. L. & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the ACM, 29*(7), 605-610.

Oliver Recruitment Group. (2003). Internet Job Index. Retrieved January 30, 2003, from http://www.olivier.com.au/jobindex.php

Parlante, N., Popyack, J., Reges, S., Weiss, S., Dexter, S., Gurwitz, C., Zachary, J. & Braught, G. (2003). Nifty assignments. *Proceedings of the 34th Technical Symposium on Computer Science Education*, Reno, Nevada, USA.

Sainsbury, M. (2003). Telstra about face as jobs go offshore. *The Australian*. All-round Country**:** 19.

Sleeman, D. (1986). The challenges of teaching computer programming. *Communications of the ACM, 29* (9), 840-841.

Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. *Communications of the ACM, 29* (9), 850-858.

Soloway, E. & Norris, C. (1999). Back to the Future. *netWorker, 3* (4), 28-29.

Biography

Michael de Raadt is a PhD student and instructor of programming at the University of Southern Queensland. Michael undertook undergraduate study at the University of Western Sydney and achieved his Bachelor of Applied Science Degree with Distinction in 1998, and was awarded First Class Honours and the UWS University Medal in 1999. Michael is also a recipient of the ACS prize for Highest Achievement.