13

Computer science education at the crossroads

Ken Robinson
Department of Software Engineering, School of Computer Science and Engineering, University of New South Wales, Sydney NSW 2052 Australia, e-mail: K.Robinson@unsw.edu.au

Abstract

Computer science has been the dominant discipline for computing education over more than four decades. This paper examines some current changes and suggests that computer science education - as it is currently presented - may be entering a period of decline as new demands assume a more prominent role. The intention of the paper is to provoke discussion on the future directions of computing education.

Keywords

Informatics, software systems, software engineering, natural sciences and engineering, informatics majors, curriculum (general), educational profiles

1 COMPUTER SCIENCE

Throughout this paper 'computer science' and 'informatics' are assumed to be interchangeable terms. Computer science is the term used most commonly in Australia, the United Kingdom, the United States of America and Canada, while the term informatics is commonly used in Europe. This paper will use the term computer science.

Since around 1960 computer training has been carried out under the umbrella of computer science. It is widely agreed that the term is more wishful thinking than grounded in any reality. Computer science courses have been generalist computing courses from which graduates entered various computing occupations and professions. At the same time it was common for electrical engineering courses to contain substantial computing content and it was common for those who wished to

Informatics in Higher Education F. Mulder & T. van Weerts (Eds.)

have significant computer hardware expertise, to undertake an electrical engineering degree combined with computer science.

Much of the discussion in this paper is drawn from experiences and initiatives at the University of New South Wales. This is done because the author is close to those experiences
and initiatives. Similar initiatives are seen to be occurring in many other universities and it is presumed that the experiences might be analogous to ours. The assumption is that what we are seeing locally might be part of a global change.

2 RECENT DEBATE: ENGINEERING VERSUS SCIENCE

In Wegner and Israel (1995) there is a discussion of the nature of computer science. In this discussion Belady (1995), Brassard (1995), Denning (1995), Freeman (1995), Hartmanis (1995a), Loui (1995), Plaice (1995), Savage (1995), Stewart (1995), Ullman (1995) and Wegner (1995) take part, prompted by the 1994 Turing Award Lecture by Hartmanis (1995b). This debate touched on the question of whether computing is an engineering or a science discipline - irrespective of what you call it. The significance of this discussion is not that it reached a particular consensus, but rather that the discussion was occurring - in the particular forum where it was occurring - at all. This discussion probably would not have happened ten years ago.

The 'engineering versus science' discussion is much older and goes back to the time when the phrase 'software engineering' was invented and coined in 1968, nearly four decades ago. While certainly not unanimously accepting the proposition that computing is an engineering discipline, the cited discussion is evidence of a gradual awareness, if not acceptance, of the notion of computing as a genuine engineering discipline. This discussion of the nature of the computing discipline is clearly related to, but is not the same as, the goals of computing education.

3 LOCAL EXPERIENCE

The strongest local evidence for a change comes from our own School of Computer Science and Engineering. This School came into being as a school in 1991, having previously been a Department in the School of Electrical Engineering since 1964. In 1989 we introduced a four-year computer engineering course, administered jointly with the School of Electrical Engineering. The objective of this course was to educate design engineers who could work on computer systems - possibly embedded - that consist of either hardware or software, or both. The new course was additional to an existing three-year computer science course.

The computer engineering course immediately established itself as one of the highest entry level computing or engineering courses in Australia and it has maintained that position. Entry into University of New South Wales courses is

based on a Tertiary Entrance Ranking - score (TER - score) which is computed from the New South Wales Higher School Certificate (NSW HSC). The HSC is a state of New South Wales public examination. A student who obtains a TER of 95 is in the top 5% in the state. The computer engineering course commenced with a minimum TER of about 97 for 60 students. In 1997 the course took in about 150 students with a minimum TER of about 92.

This year, 1997, we introduced a new four-year software engineering course administered jointly with the School of Information Systems. The software engineering course commenced with an intake of 25 students with a minimum TER of 97.

Since 1989 the computer science course has moved from being our major computing course - in terms of numbers of students - to being second to computer engineering. Over the same period the minimum entrance requirement has fallen to a TER of 78 in 1997.

Anecdotal evidence from other Australian universities suggests that interest in and entrance levels for computer science courses have been falling in recent years. Anecdotal evidence from the United Kingdom and other countries suggests that there are similar experiences in those countries.

4 WHAT IS IN A NAME?

Computer engineering and software engineering form a continuum from the electrical and computing hardware interface through software to the application interface. These courses are intended to satisfy a demand from society and industry for system developers, whether the systems are implemented in software or hardware. If the courses are well targeted and successful - it is too early to judge in the case of software engineering - then they will be satisfying a need previously supplied by computer science. We would expect to see both of the engineering courses having large numbers of very good students and presumably a much smaller number of students doing computer science.

There is a serious question raised by the above expectation. If the new engineering courses become the preferred source of graduates for industry and commerce, why would anyone want to study a computer science course? That is not intended to be a rhetorical question. It is expected that there may be cogent reasons, but that they are not likely to include many of the dominant reasons that have enticed students to computer science in the past.

It could be thought that the move from computer science to engineering courses is merely a change of name. Has anything of substance occurred? It is true that the new courses share a substantial number of subjects in common with the computer science course. Indeed it is almost the case that the new courses contain the computer science course. But there are new subjects and there is a different emphasis. The engineering courses have a much greater concern for the

development of systems: hardware systems, software systems and combined hardware and software systems.

Characteristics of computer science courses

The characteristics that appear to distinguish the objectives of computer science courses are as follows.

- Computer science is devoted to developing 'computing in the small'.
- Despite a significant level of theory in computer science courses, that theory does not address design and implementation. This is partly another manifestation of the 'computing in the small' characteristic, but it also derives from a concentration on algorithms and data structures, and a lack of concern with systems.
- Computer science courses do not develop a strong concern for reliability and quality. Indeed it might be claimed that computer science tends to perpetuate the hacker approach to system development.
- Computer science courses seem to have no particular focus.

Characteristics of engineering courses

As a counterpoint to the above an engineering course has to produce graduates who know how to develop (potentially large) systems. This requires:

- an appreciation of the problems which occur in developing systems;
- methods for aiding the design and implementation of systems;
- an understanding of the process of system development;
- a concern for reliability and quality, which entails a sense of professional responsibility that is characteristic for the engineering professions.

While emphasizing the above aspects of an engineering education, it is desirable that innovation - a strong characteristic of computer science - is not lost. It is also important to emphasize that the above engineering requirements should not be achieved by mixing common computer science subjects with qualitative subjects. There is a need to answer the requirements with a course that is as strongly based on sound principles and as rigorous as possible.

5 INVERTING THE CURRICULUM

Evidence of 'computing in the small' can be seen in traditional computer science course sequences: the course will commence with programming in the small, proceed to intermediate data structures and algorithms, and then move on to various application or problem domains.

There is an argument for effectively inverting this sequence. The discussion of algorithms and data structures is not well motivated, since the need for particular data structures or algorithms is driven by particular requirements, and hence is problem and implementation dependent. It would be much more supportive of

developing an understanding of systems to start with nontrivial systems for which students developed components, starting from simple components and working towards complete modules. Particular requirements would provide an opportunity to discuss alternative data structures and algorithms. Something like this approach is advocated in various object-oriented design subjects. It certainly fits in well with a comprehensive software engineering approach to the development of systems.

A trivial, but significant, example of the consequence of the 'programming in the small' attitude can be seen in the selection of the first programming examples. A tradition has developed in which the canonical first program is a program that writes 'hello world' on a text file. The next program probably reads numbers from an input file and writes the value of some expression on output. If these exercises are attempted using a modern object-oriented programming language, the exercise may be not particularly simple, needing knowledge of classes and possibly exceptions.

Based on the assumption that these programs are intrinsically simple and fundamental, this might be used as an argument for not using such programming languages. However, if the first programs are given in the context of a complete system, then maybe the first exercise is a procedure/function/method that receives values as parameters and returns a result - perhaps the circumference of a triangle. Starting from this simple exercise there are many procedures which can be developed for a nontrivial system; for example, a simple simulation of an Automatic Teller Machine (ATM) with all input/output through a supplied graphical interface. Such exercises would develop, from the beginning, an understanding of system requirements and specification.

Examples of such an inverted curriculum - albeit for specific object-oriented programming languages - can be found in Meyer (1993) and Duke (1997).

6 CONSEQUENCES FOR COMPUTER SCIENCE

If we assume that computer and software engineering courses are successful, both in attracting students and in satisfying many of the needs of industry, then we have to concede that computer science courses will lose a significant vocational component from their list of attractions. This will be true at all institutions, whether a particular institution has engineering courses or not. The competition will be both within institutions and between institutions. The serious question then is: 'What happens to computer science?'

One could think of the following possibilities.

1. Computer science could become irrelevant and dwindle to a very small size, perhaps disappearing in some institutions.
2. Computer science could be used mainly in conjunction with major studies in other disciplines.
3. Computer science could fashion itself as a true science of computing for the first time. Computer science would then be concerned with the more

foundational elements of computing and be closely associated with new research directions in fundamental aspects of computing.

All of the above probably represents a significant contraction in the size of computer science. Option 1 is unattractive, and a mixture of 2 and 3 seems most desirable. It is important to appreciate that option 3 will not happen without serious rethinking.

7 CONCLUSION

If local experiences are representative of global ones, then we may be seeing a transition period for computing education. We may be experiencing the emergence of computing from a craft period to an engineering period. At least society may have reached a point where it now requires graduates who are able to develop the complex computing systems on which society is becoming more dependent.

We would argue that conventional computer science courses are not well suited to producing such graduates and that there is a need for courses with a strong engineering orientation. The important point is not the name of such courses, but their objectives and hence their content. Such courses could be named 'computer science', but because of the significant shift in objectives it is likely that such courses will be called 'engineering' courses. This would leave computer science and computer/software engineering operating in parallel, and strongly suggests that the objectives of computer science courses will need to be redefined. Hopefully this paper contributes to that.

8 REFERENCES

Duke, R. (1997) In search of the inverse curriculum, in *The Proceedings of the Second Australasian Conference on Computer Science Education,* July 1997, 65-70.

Hartmanis, J. (1995b) Turing Award Lecture: on computational complexity and the nature of computer science. *ACM Computing Surveys*, **27** (1), 7-16.

Meyer, B. (1993) Towards an object-oriented curriculum. *Journal of Object-Oriented Programming,* May 1993, 76-81.

Wegner, P. and Israel, M. (1995) Computing Surveys symposium on computational complexity and the nature of computer science. *ACM Computing Surveys,* **27** (1), with the following contributions:

- Belady, L.A. (1995) The disappearance of the 'pure' software industry, 17-18.
- Brassard, G. (1995) Time for another paradigm shift, 19-21.
- Denning, P.J. (1995) Can there be a science of information?, 23-24.
- Freeman, P.A. (1995) Effective computer science, 27-29.

- Hartmanis, J. (1995a) Response to the essays 'On computational complexity and the nature of computer science', 59-51.
- Loui, M.C. (1995) Computer science is a new engineering discipline, 31-32.
- Plaice, J. (1995) Computer science is an experimental science, 33.
- Savage, J.E. (1995) Will computer science become irrelevant?, 35-37.
- Stewart, N.F. (1995) Science and computer science, 39-41.
- Ullman, J.D. (1995) The role of theory today, 43-44.
- Wegner, P. (1995) Interaction as a basis for empirical computer science, 45-48.

9 BIOGRAPHY

Ken Robinson is a senior lecturer and head of the Department of Software Engineering in the School of Computer Science and Engineering at the University of New South Wales (UNSW). He has a long record of teaching computer science and software engineering with a special interest in rigorous software development methods, and in the recognition of computing as an engineering discipline. He played a leading role in the introduction of a new computer engineering course in 1989 and a new software engineering course in 1997 at UNSW.