8

Teaching introductory computer science as science of information

Gopal K. Gupta
Department of Computer Science, James Cook University
Townsville, Qld 4811, Australia, e-mail: gopal@cs.jcu.edu.au

Abstract

Since the development of first computer science programs in the early 1960's, introductory computer science teaching continues to be problematic with many departments reporting high drop-out and high failure rates in their introductory courses. The present approach to teaching introductory computer science as teaching procedural programming using an apprenticeship approach needs to change. We suggest an alternate approach which is based on information, its processing, presentation and communication. Programming continues to be an important part of the proposed curriculum, but it does not occupy the central place which it currently does.

Keywords

Informatics, information systems, informatics majors, curriculum (general), curriculum (start)

1 INTRODUCTORY COMPUTER SCIENCE COURSES

The computer science curriculum has been a topic of intense discussion since the birth of the discipline in the early 1960's, as is for example reflected by the debate in Denning (1989). A number of model curricula, including Curriculum 68, Curriculum 78 and Curriculum 91, have been developed by ACM and have been widely used as basis for curriculum design. Other computing societies have developed their own recommendations. Introductory computer science courses appear to have evolved only slowly over the last thirty years. Curriculum 68 noted that the first course (Course B1: Introduction to Computing) was designed to

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

provide the student with the basic knowledge and experience necessary to use computers effectively in the solution of problems and suggested that the course could be used both for majors as well as nonmajors.

More recently, Koffman, Miller and Wardle (1984) presented a model curriculum for an introductory course CS1 with the following objectives:

- to introduce a disciplined approach to problem-solving methods and algorithm development;
- to introduce procedural and data abstraction;
- to teach program design, coding, debugging, testing and documentation using good programming style;
- to teach a block-structured high-level programming language;
- to provide a familiarity with the evolution of computer hardware and software technology;
- to provide a foundation for further studies in computer science.

Although most introductory computer science courses, we suspect, have objectives similar to those listed above, the debate about curriculum for the introductory course continues unabated. For example, there has been considerable discussion in the literature whether an introductory course should focus narrowly on some aspect of computer science (often programming) and ignore breadth of coverage, or be a breadth-first course which gives up some of the narrow programming emphasis. Often though the first one or two courses are dominated by programming. Scragg, Baldwin and Kooment (1994), and Doran and Langan (1995) present what in their view are symptoms of the problems with introductory computer science courses. These are high drop-out rates in the undergraduate programs, in particular in the introductory courses, complaints by the employers that the graduates are not able to apply what they have learned and high drop-out rates from computer science Ph.D. programs (which presumably reflects the poor preparation). Furthermore, anecdotal evidence suggests that computer science students completing an introductory course sometimes do not even know how to use a spreadsheet effectively. Although these symptoms are by no means found in all computer science programs, these are common enough to be familiar to most computer science academics. Bagert, Marcy and Calloni (1995) give an extreme example of drop-out and failure rates in which a first year class of 216 students was reduced to only five graduates four years later. A number of solutions has been proposed to solve these problems. Some suggest, for example, that an introductory course should include a study of good examples of software systems, modify and combine programs as well as creating them and present theory and models in the context of practice. Baldwin *et al.* (1994) and Scragg *et al.*(1994) state that the central mission of their introductory sequence is to teach design, theory and empirical analysis. Doran and Langan (1995) take a very different approach. They refer to the six levels of learning in the educational process: knowledge, comprehension, application, analysis, synthesis and judgement. They comment that the introductory courses should try to primarily

teach the first three levels, since the other three (analysis, synthesis and judgement) require a good mastery of the first three and maturity gained by extended usage.

Most conventional introductory courses expect students to complete programming assignments without a great deal of assistance, but this is quite unrealistic since programming is a complex cognitive task. The apprenticeship approach, although suitable for learning a trade or technical skills, is perhaps not suitable for learning conceptual material and in any case is often ineffective due to lack of staff resources. It results in average students constantly battling with programming assignments and thus having no time to consider the conceptual basis of the discipline. The concept of closed laboratories, which has not been common in computer science teaching in North America, but has been the normal practice in Australia for many years, helps, but does not overcome the problems. Given the conventional curriculum many students think that computer science is just programming; that is what we tell the students in our introductory courses as noted by Denning *et al.* (1989): 'The emphasis on programming arises from our long-standing belief that programming languages are excellent vehicles for gaining access to the rest of the field, a belief that limits our ability to speak about the discipline in terms that reveal its full breadth and richness. ..Clearly programming is part of the standard practices of the discipline and every computing major should achieve competence in it. This does not, however, imply that the curriculum should be based on programming or that the introductory courses should be programming courses'.

Furthermore, the conventional introductory curriculum is too dependent on changes in technology which an introductory course ought not to be. The curriculum discussions in computer science departments are thus often dominated by whether the programming language being used needs to be changed to the latest one to the neglect of more important issues. Such language discussions have been known to turn into religious wars which can do serious damage to the fabric of a department.

To summarize, we believe that current introductory computer science courses suffer from the following problems.

- The courses focus on programming and algorithmic computation and do not present a broad picture of computer science.
- The courses require too much programming in too early a stage which is often intimidating for at least a significant minority of students, perhaps more so for the female students.
- The courses use an apprenticeship approach which is often very time consuming for most students given the lack of staff resources.
- The pass rates in most current introductory courses are often low and the drop-out rates high; retention of less than 50% after the first course appears common but is in our view unacceptable.
- Although the programming assignments can be time-consuming, the courses convey few intellectual challenges to some of the brightest students.

Current trends in curriculum design appear to indicate that new developments are unlikely to solve the problems listed above. Nearly the only concern in designing new introductory curricula presently appears to be object-oriented programming, although it is clear that use of object-oriented programming is not going to solve the problems identified above.

2 NEW APPROACH

We believe that a different approach to teaching computer science is needed, not only because the present approach suffers from significant difficulties, but also because computing and the computing industry have changed dramatically over the last four decades. The primary concern in computing now appears to be shifting to information storage, retrieval, display and presentation. As a result the information technology industry now offers many opportunities for people with a variety of skills (e.g. networking, Web applications as well as programming; see Keaton and Hamilton, 1996). We therefore need a new curriculum which recognizes that not all students studying computer science are going to (or need to) become programmers or software developers. Some may also not be capable of and/or interested in acquiring such skills beyond basic programming. We should not deny such people an opportunity to develop computing knowledge and expertise in areas other than software development. In some ways the computer science academic community faces a real choice now: We may design computer science curricula to be inclusive, so that students from many different backgrounds may pursue computer science studies, or we may design curricula to be exclusively focused on producing first-rate software developers. The exclusive option is not the one we recommend and in our view exclusive curricula proposals like those of Dijkstra (Denning, 1989) and Parnas (1990) are not worth considering since they propose solutions which are likely to lead to a serious decline of the discipline. We propose that a very broad view of information be the basis of the computer science curriculum and we agree with Denning (1995) that a computer science program should be taught as the science of information. However, what we propose is very different from most 'information systems' and 'information science' curricula. Information systems curricula often only consider a limited view of information, in particular the role of information in decision making. Information science curricula on the other hand often are closely related to 'library science'. Our proposed curriculum takes a very broad view of information as the basis of computer science and focuses on the concept of information and the manipulation, communication and display of its representations, rather than on algorithmic computation. Following this approach the introductory computer science course, as well as the courses which follow, are significantly different from what we teach today. As an example we propose that an introductory course consists of the following topics.

The concept of information

- the concept of information and its many forms, the question 'what is information?', definition and characteristics of information;
- non-temporal information:
  - > linear and non-linear **text**:
    - ~ linear text without form, sequence of characters, how many different characters, ASCII, ISO character sets, character sets for LOTE (Languages other than English); text with form, need to store content and form both, storing form by using mark-up languages (Troff, Latex, SGML, ODA), storing form by specifying form in a WYSIWIG-editor, presentation, fonts, device independent fonts, storage and printing of fonts, geometric descriptions of fonts, kerning, PostScript;
    - ~ non-linear form of text: hypertext, representation of hypertext, Web;
    - ~ operations on text: retrieval, character and string operations, editing, formatting, pattern matching and searching, spell checking, style checking, compression, encryption;
  - > **images**:
    - ~ images as a two-dimensional array of pixels, monochrome, grey and colour images; colour models, representation of images;
    - ~ operations on images: editing, point operations, filtering, compositing, geometric transformations, conversions, compression;
    - > **graphics**:
    - ~ difference between graphics and image data, representation of graphics data, geometric modelling (GKS, PHIGS, etc.), solid models, other models;
    - ~ operations on graphics data: editing, shading, mapping, lighting, viewing, rendering;
- temporal information:
  - > video:
    - ~ video as sequence of images or frames, analogue video and digital video, analogue representation and major formats (NTSC, PAL, SECAM, etc.), video storage, digital representation, data rates, digital video storage;
    - ~ operations on video: video sources and sinks, video mixer, retrieval, editing, compression of digital video, MPEG, JPEG, etc.;
  - > audio:
    - ~ digital and analogue audio representation, speech, encoding, audio formats (CD, DAT, etc.);
    - ~ operations on audio: storage, retrieval, editing and filtering;
  - > music:
    - ~ difference between music and audio, representing music (MIDI, SMDL);
    - ~ operations on music: playback, synthesis, editing, composition;
  - > animation:

  - ~ animation as sequence of synthetic image frames, animation versus video, animation models;
  - ~ operations on animation: motion and parameter control, rendering;
- information transformation:
  - > transforming text to audio, speech to text, music to audio, animation to video, etc.;
- optional:
  - > what is knowledge? representing knowledge e.g. plans, games, rules, etc.;
  - > natural language representation and understanding.

Information storage and retrieval

Information storage and retrieval, value of information, need for many different types of manipulation, etc. Storage and retrieval of large amounts of information on paper; books, chapters, table of contents, index, telephone directory, organisation of books in library, files and filing cabinets in a office, inventories, archiving, introduction of terms like sequential, index sequential, trees; analogies between manual information handling and information handling by computers. Information as a commodity and its unique properties, needs of graphical display of information, computing aggregate values, and other manipulations.

Computers as machines

Computers as (simple) machines which manipulate information; simple introduction to computer organization. Conceptual model of a computer, computer examples: computer games, ATM, a computer in a washing machine or a car, large mainframes, desktop, laptop and mobile computers. Concept of instructing computers; analogies between instructing a computer and instructing a human worker.

Input and output

The need to input information to computers and to output information from computers; many different types of inputs and outputs. Input to computers via keyboard, scanners, cameras, speech, instruments, touch screens, pen, mouse, joystick, etc.; conceptual understanding of how these devices work, devices like scanners and fax and their working; importance of user-friendly information input. Output from computers via monitors, printers of different types, speech, music, graphical output, control signals, etc.; conceptual understanding of how these devices work, importance of user-friendly information output.

Information storage

The need to store information in computers, techniques for storing and retrieving information; introduction to storage devices: core memory, disks, CD-ROMS, tapes etc.; introduction to data structures (sequential, indexed sequential, trees, hashing), concept of relational database and a simple retrieval language.

Ownership
The concept of ownership, availability and fair use of information; issues of copyright, personal privacy, information rich and information poor, ethics; who owns information, access to information held by public and private organisations, intellectual property rights, concept of information privacy, information privacy principles, information society, concept of information rich and poor societies, ethics for information processing professionals.

Communication
The need to communicate information from an input device to a computer, from a computer to an output device and from one computer to another. Computer communications, e-mail, Internet, World Wide Web, networks, mobile computing, conceptual understanding of how they work.

Manipulation
The need to manipulate information (mathematical computations, symbolic computations, intelligence), techniques for simple and complex information manipulation; information manipulation via available packages, word processing, spreadsheets, graphics packages, mathematical packages, expert systems; examples where a package will not do the job.

Instructions
The need to instruct the machine to manipulate information; introduction to one procedural language; examples of algorithms and their implementations using the language, debugging, testing and documentation; procedural and non-procedural languages, machine code, assembler, components of a procedural language, introduction to one procedural language, examples of algorithms, simple programs implementing those algorithms, extension of those algorithms and programs, the importance of debugging, testing, and documenting software.

3 CONCLUSION

The dramatic changes in computing and the computing industry in the last thirty years require that we look at computer science curricula afresh. In this paper we have considered the introductory computer science curriculum and have presented a curriculum which is significantly different from any proposal we know. In our view the proposed curriculum is exciting, since it approaches computer science from an information point of view which enables a much broader approach to computer science in the first course. The new approach has a number of other advantages. Major programming is deferred to and isolated in the design and implementation group project courses which should follow the introductory course (for details see Gupta, 1996a). The primary focus here is on implementation and we believe that sufficient staff resources ought to be provided to guide the students to good

implementations. The new approach is believed to reduce the frustration and workload in the first course allowing the student to focus on the conceptual basis of the discipline.

4 ACKNOWLEDGEMENTS

A number of people have helped me in putting these ideas together. It is a pleasure to thank Reinhart Gillner, Shyam Kapur, Bruce Litow, Tony Sloane, Rodney Topor, Chris Wallace, and David Wessels. An earlier version of this paper was presented at CSI96 conference in Banglore, India (Gupta, 1996b).

5 REFERENCES

Bagert, D., Marcy, W.M. and Calloni, B.A. (1995) A Successful Five-Year Experiment with a Breadth-First Introductory Course. *SIGCSE Bulletin*, **27** (1), 116-120.

Baldwin, D., Scragg, G. and Kooment, H. (1994) A Three-Fold Introduction to Computer Science. *SIGCSE Bulletin*, **26** (1), 290-294.

Denning, P.J. [ed.] (1989) Teaching Computer Science. *Communications of the ACM*, **32** (12), 1397-1414.

Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A.B., Turner, A.J. and Young, P.R. (1989) Computing as a Discipline. *Communications of the ACM*, **32** (1), 9-23.

Denning, P.J. (1995) Can There be a Science of Information? *ACM Computing Surveys*, **27** (1), 23-25.

Doran, M.V. and Langan, D.D. (1995) A Cognitive-Based Approach to Introductory Computer Science Courses: Lessons Learned. *SIGCSE Bulletin*, **27** (1), 218-222.

Gupta, G.K. (1996a) *Teaching Computer Science as the Science of Information.* Internal report TR11, Dept of Computer Science, James Cook University.

Gupta, G.K. (1996b) *Teaching Introductory Computer Science: Past, Present and Future*. Proceedings of CSI96, Tata McGraw Hill, 95-102.

Keaton, J. and Hamilton S. (1996) Employment 2005: Boom or Bust for Computer Professionals. IEEE Computer, **29** (5), 87-98.

Koffman, E.P., Miller, P.L. and Wardle, C.E. (1984) Recommended Curriculum for CS1: 1984 a report of the ACM Curriculum task force for CS1. *Communications of the ACM*, **27** (10), 998-1001.

Parnas, D.L. (1990) Education for Computing Professionals. *IEEE Computer*, Jan 1990, 17-22.

Scragg, G., Baldwin, D. and Kooment, H. (1994) Computer Science Needs an Insight-Based Curriculum. *SIGCSE Bulletin*, **26** (1), 150-154.

6 BIOGRAPHY

Gopal Gupta is professor and head of the Department of Computer Science at James Cook University since 1986. He graduated in 1965 from the University of Roorkee (India) and received in 1968 a Masters degree in engineering from the University of Manitoba (Canada). After working at Bristol Aerospace in Winnipeg he took up a studies in computer science and completed a Masters degree at the University of Waterloo (Canada). From 1971 he spent 14 years at the Monash University (Australia). In 1976 he received his Ph.D. from this university. His last position at the Monash University was senior lecturer and deputy chairman of the Computer Science Department. During the Monash period he has been on leave at the University of Illinois (1 year) and at the Asian Institute of Technology in Bangkok (2 years). His research interests are: data structures, database systems and computer science education.