PECTOPAH: Promoting Education in Computer Technology using an Open-ended Pedagogically Adaptable Hierarchy

Hugh Osborne, Shirley Crossley and Jiří Mencák
School of Computing & Mathematics
University of Huddersfield
Huddersfield HD1 3DH, U.K.
{h.r.osborne,j.mencak}@hud.ac.uk
shirleycrossley@blueyonder.co.uk

William Yurcik
Dept. of Applied Computer Science
Illinois State University
Normal
Illinois
USA
wjyurci@ilstu.edu

1. TEACHING COMPUTER ARCHITECTURE

1.1 Computer Systems Architecture

An understanding of Computer Systems Architecture (CSA) is essential to an understanding of Computer Science. There is however a tendency, at all levels, in teaching Information and Communications Technology (ICT) to neglect CSA, but teaching ICT without teaching CSA is like teaching Russian without teaching the Cyrillic alphabet — students may become reasonably fluent in the application of abstract high level skills (e.g. they know that the Russian for restaurant is *restoran*), but lack the basic skills needed to maintain and extend those skills (e.g. they cannot identify PECTOPAH as being the "real" Russian for restaurant). There are two major reasons for the neglect of CSA in teaching ICT. There is a misconception of the effect of technological change, and there is a tendency to use inappropriate didactic tools.

1.2 The Rôle of Technological Change

"With IT technology developing so rapidly, is it really worth trying to teach something that will be out of date within a very short time?" As rapid as the developments have been it is the high level applications of IT that have changed — *the basic principles of CSA are essentially the same as they were 50 years ago.* Learning these basic principles allows students to build their understanding of high level IT applications on their knowledge of the basic principles of digital computers, confident that these principles are unlikely to change quickly and that they will be able to apply the same understanding to further developments and thus maintain a state of the art knowledge.

1.3 Using the Right Tools

CSA is traditionally considered to have a high learning threshold. This is due to the difficulty of teaching it in an incremental *and* hands-on fashion. For CSA hands-on experience should be provided by writing low level programmes, but low level programming already requires a solid grounding in aspects of CSA, e.g. internal data representation, the memory hierarchy, interaction with peripheral devices, etc.

This perceived difficulty is again a misconception, due to the use of inappropriate tools. A common approach is to use the inbuilt assembly language of some real machine to provide hands-on experience. These languages are not designed as didactic tools, but as programming tools for experienced users who already have a thorough understanding of CSA, making such languages hard to understand, and making it hard for students to separate the (manufacturer specific) incidental from the (subject wide) essential. The frequently cryptic documentation only exacerbates the problems students have. Using such a tool to learn CSA is akin to trying to learn Russian using only a Russian/English-English/Russian dictionary — a very useful tool in skilled hands, but inappropriate as a beginner's guide.

This abstract describes three tools that together provide a progressive hierarchy of teaching aids thta can be used at many levels of teaching, providing students with a seamless incremental toolbox that can be used throughout their education.

The remainder of this abstract is organised as follows: Section 2 describes the teaching philosophy behind the toolbox; in Section 3 the three components of the toolbox are described; Section 4 discusses the use of these tools; and section 5 is a summary of their integration in the toolbox.

2. INCREMENTAL TEACHING

The aim of any good course must be to introduce students to new concepts in an incremental fashion. The subject matter must be analysed and a plan of delivery developed so that the *learning threshold* is at all times as low as possible. Learning of concepts is strongly reinforced by "hands-on" experience, and this should be introduced as early as possible — any course in which many weeks of background introduction are required before students can undertake realistic exercises is likely to be perceived as "difficult" or "too theoretical".

Low learning thresholds by no means exclude high learning expectations. It is essential that the teacher has high expectations of the students' learning, and communicates these to the students. Students perform to expectations, so high expectations bring out the best in achievement.

These two aims are strongly related to the need to structure courses so as to enable students of the widest possible range of abilities to profit to the maximum of their capabilities from the material on offer. A well structured course will provide students with knowledge and skills at various levels. The course should contain enough advanced material to challenge the more able student, allowing them the opportunity to develop and prove their ability, while ensuring that the basics are covered in sufficient detail for the less able to provide them with the basic knowledge expected of them.

Incremental teaching is also the ideal on a longer timescale. Students making the transition from, for example, secondary to tertiary education often experience a "fault line" where there is a mismatch between their prior knowledge and experience and the prerequisites assumed by their new institution. While such problems are to some extent unavoidable, the development of national curricula and the provision of integrated tools and methodologies can help to alleviate them.

3. APPROPRIATE TOOLS FOR CSA EDUCATION

3.1 Primary and Secondary Education — "How Computers Really Work"

There is a shortage of appropriate material for teaching late elementary and secondary school pupils the essentials of Computer Systems Architecture. "How Computers Really Work" is a pilot interactive CD-ROM for teaching CAS to primary and seondary school children. Students are guided throughout by "Chip", an animated floppy disk. There are areas on the CD covering peripherals, computer hardware and software, the internet and the history of computing. There is also a quiz consisting of multiple choice questions covering material from all other sections of the CD. The largest area of the CD is Computer Architecture area dealing with the CPU, memory and data. The description of the CPU is based on the Postroom Computer (see section 3.2).

3.2 Introductory Undergraduate Level — "The Postroom Computer"

The problem of teaching low level programming at an introductory undergraduate level was addressed as early as 1965 by Stuart Madnick and John Donovan (see e.g. [3]). In the *Little Man Computer* (LMC) they provided an extremely simplified model of low level programming and computer architecture. The LMC model has proven to be of lasting popularity, as the number of LMC emulation programmes currently in existence, 35 years after it was originally proposed, shows (see, e.g., [11] or [10] for a survey).

The *Postroom Computer* (PC) [8, 9] is an extended emulation of the LMC model, in which the emphasis is on flexibility and generality. It is designed to introduce aspects of CSA and low-level programming in an incremental way. The extensions are designed to provide a range of computing models within the LMC/PC paradigm. As they are introduced they can be related both to the LMC/PC paradigm and to "real" machines. The PC provides a powerful and flexible tool for teaching CSA. By adding orthogonal extensions to Madnick and Donovan's basic LMC aspects of CSA can be introduced in a stepwise fashion, never overwhelming students with details, yet leading eventually to a full understanding of the principles of CSA.

The PC also introduces students to a more formal description (*Update Plans*, see section 3.3) of computer systems architectures.

3.3 Advanced Undergraduate/Postgraduate — "Update Plans"

Update Plans (UP) is a formalism for the description of abstract machines and algorithms. UP is particularly suitable as a specification language for the description of large classes of computer systems architectures.

UP has didactic uses in many areas of Computer Science other than CSA — for example data structures and compiler construction. The common denominator in all these applications is that of a *pointer*. Pointers are intrinsic to Computer Science. Each field of Computer Science seems to use its own more or less ad hoc notation for describing pointers and operations on pointers, thus impeding crossover of students' skills from one area to another. UP is a "universal" pointer specification language. Consistent use of UP as a supplement to the traditional notations can greatly enhance students' ability to apply skills learned in one domain to other pointer applications.

4. USING THE TOOLS

4.1 How Computers Really Work

A pilot version of the interactive CD-ROM was tested on a group of children and their teachers. The children all enjoyed the package and liked the classroom metaphor used in the CD-ROM. The teachers found it bright and cheerful, and felt that the animations would catch the children's attention. The children did have some difficulty in understanding the Postroom Computer section. However, in the pilot version, there was no facility for hands-on programming of the Postroom Computer. Fully integrating a user friendly GUI to the Postroom Computer would undoubtedly greatly enhance the usefulness of this tool.

4.2 The Postroom Computer

The PC is supported by both online documentation[1] and a fully integrated system of course materials [2], including a range of exercises allowing students of all levels to advance their knowledge and skills.

Experience has shown that after one semester (12 weeks), with one hour of lectures and one hour of supervised practical exercises per week, first year first semester undergraduates with no prior knowledge of the subject demonstrate a good understanding of the subject matter and can successfully undertake a range of ambitious low level programming exercises of a level normally considered to be too advanced for introductory CSA courses. The students typically achieve a higher level of understanding of the principles of CSA than usual at this level of instruction.

4.3 Update Plans

UP was not originally developed as a teaching aid, but as a theoretical tool. It has been the target of academic research [4, 5, 6, 7], which has shown that the formalism is a valuable aid for the description and analysis of a wide range of computer systems architectures. UP has also been used as a teaching aid at various universities, and has shown its worth in helping students to understand how the components of computers, both hardware and software, interact. An implementation of a subset of Update Plans is currently being used as a teaching aid for an advanced course on the implementation of functional languages.

5. THE TOOLBOX

Each of the three components described is a useful tool in its own right. Together they provide a long term incremental tool for teaching CSA.

How Computers Really Work provides a basic introduction to CSA, including a first contact with the Postroom Computer model.

The Postroom Computer can be used as an incremental hands-on teaching aid for introductory undergraduate CSA. No prior exposure of the students to the Postroom Computer is required for successful deployment of this tool, though it would of course be an advantage. As well as introducing students to the important concepts of CSA, the Postroom Computer model will also probably be their first introduction to precise and formal descriptions of computer systems structures, using Update Plans.

Update Plans provide a tool for further development of student's understanding of CSA, allowing them to develop and test their own models of advanced computer systems architectures. Update Plans are also applicable to many other areas of computer science (e.g. data structures and compiler construction), thus facilitating crossover of students' skills and understanding. Further applications of Update Plans can be developed to encourage this crossover, and to provide an open-ended tool in teaching computer science.

6. REFERENCES

[1] `http://scom.hud.ac.uk/staff/scomhro/Courses/PostroomComputer/`.

[2] `http://scom.hud.ac.uk/staff/scomhro/Courses/CFS155/`.

[3] Irv Englander. *The Architecture of Computer Hardware and Systems Software*. John Wiley & Sons, New York, 2000. Second edition.

[4] Hugh Osborne. The semantics and syntax of update schemes. In *Code Generation — Concepts, Tools, Techniques*, Workshops in Computing. Springer Verlag, 1992.

[5] Hugh Osborne. Update Plans. In *Proceedings of the 25th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1992.

[6] Hugh Osborne. *Update Plans — A High Level Low Level Specification Language*. PhD thesis, University of Nijmegen, 1994. `http://scom.hud.ac.uk/staff/scomhro/Papers/PhD/phd.html`.

[7] Hugh Osborne. Update Plans for parallel architectures. In M. Kara, J.R. Davy, D. Goodeve, and J. Nash, editors, *Abstract Machine Models for Parallel and Distributed Computing*. IOS Press, 1996.

[8] Hugh Osborne. The Postroom Computer. *ACM Journal of Educational Resources in Computing*, 2(1), March 2002.

[9] Hugh Osborne. The Postroom Computer: Teaching introductory undergraduate computer architecture. In *Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002)*, 2002.

[10] Gregory S. Wolfe, William Yurcik, Hugh Osborne, and Mark Holliday. Teaching computer organization/architecture with limited resources using simulators. In *Proceedings of the 33rd ACM Technical Symposium on Computer Science Education (SIGCSE 2002)*, 2002.

[11] William Yurcik and Hugh Osborne. A crowd of Little Man Computers: Visual computer simulator teaching tools. In *Proceedings of 2001 Winter Simulation Conference*, New York, 2001. ACM.