Too Much Material, Too Little Time: The computer education curriculum dilemma

Rick Duley, S P Maj, A Boyanich
Edith Cowan University, Perth, Western Australia

Abstract

Adding an engineering component into an already overcrowded computer science course to create a course suitable for the graduation of a potential professional Software Engineer (SE) reminds people of the problem of getting a quart into a pint pot — there is far too much material and far too little time. Since Curriculum '68 was published, designers of computer science curricula have been on the horns of a dilemma — what to leave out! If one is to cover the requisite knowledge areas and yet reduce the amount of information which has to be transmitted then the obvious solution is to reduce the depth in which that information is studied. The question often asked is whether such a reduction is acceptable in software engineering education. In the authors view the real question is whether or not a software engineering student actually needs the knowledge of the inner workings of a VLSI chip, CD-ROM, a Zip Drive or Modem which is appropriate to a Computer Scientist or Computer Systems Engineer. We support the case that it is not since an SE is a user of computers rather than a maker of them. Abstraction provides an effective means of reducing the volume of information which has to be transmitted in the classroom while allowing the students to formulate an adequate conceptual model of the content of the knowledge area. This paper introduces the classroom-proven concept of B-Nodes which present each device within a PC (microprocessor, hard disc drive etc.) as a data source/sink capable, to various degrees, of data storage, processing and transmission. Independent of architectural detail, experimental work to date has demonstrated that this model can accommodate rapid changes in technology, avoiding time-consuming transmission of low level detail while maintaining conceptual integrity

1. Volume of material — the perennial problem

Since Curriculum '68 (CC'68) was published, designers of Computer Science (CS) curricula have been on the horns of a dilemma — what to leave out! CC'68 grouped the subject areas of CS into three divisions: 'information structures and processes', 'information processing systems' and 'methodologies'. These were further subdivided into a total of 17 subjects which were, in turn, further subdivided into 82 individual topics. Then, in discussing 'Related Areas'

commented that the list was *'somewhat restricted'* because it was not feasible to list all the areas which might be related to a CS curriculum[1]. Under the headings *'Mathematical Sciences'* and *'Physical and Engineering Sciences'*, the report then listed a further 21 areas of related knowledge running from elementary analysis to quantum mechanics. By 2000, some of the subject areas covered by CC'68 had been dropped (e.g. text processing — including justification, the design of concordances and applied linguistic analysis) and yet the Draft Version of Computing Curricula 2001 (CC'01), in what it described as a *'Tentative list of topics in the computer science body of knowledge'*, listed 14 knowledge areas subdivided into 120 topics of which 62 were listed as core[2].

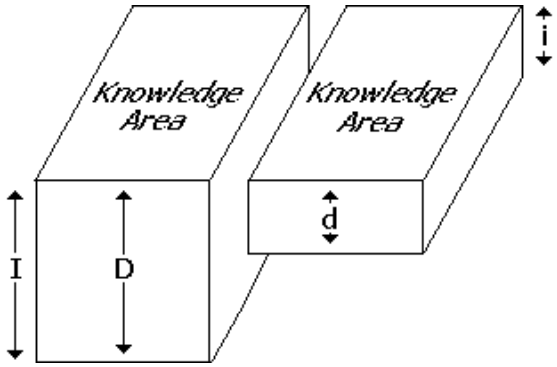

Figure 1 : Depth Reduction

In Australia, the Institute of Engineers, Australia, (IEAust) in its discussion paper on Software Engineering (SE) as a Professional Engineering Discipline published in March of 1999, listed 15 specific areas of expertise related to an accreditable SE degree[3]. These were expanded in Appendix B into a sample course outline which completely fills a three-year undergraduate course. Given that six of the 11 accredited courses are baccalaureates of engineering (in SE) which tend to be 50% traditional engineering and to involve only an extra six months of academic study, it can be seen that the volume problem persists in that, while the knowledge base expands rapidly, only a fraction of it can be taught in a student's time at university[4].

If one is to cover the requisite knowledge areas and yet reduce the amount of information which has to be transmitted (I→i) then the obvious solution is to reduce the depth in which that information is studied (D→d). The question often asked is whether such a reduction is acceptable in SE education. In the authors' view the real question is whether or not a SE student actually needs the knowledge of the inner workings of a VLSI chip, CD-ROM, a Zip Drive or Modem which is appropriate to a Computer Scientist or Computer Systems Engineer. We support the case that it is not since an SE is a user of computers rather than a maker of them and that students need to be familiar with the tools of the trade at the user (not designer) level[5]. Therefore, we believe that a conceptual understanding of such devices would adequately serve — at least in the initial stages. This attitude is based on a constructivist approach to the cognitive process.

Constructivism is a theory of learning. It holds that students construct knowledge rather than merely receive it. It has been extremely influential in science and mathematics and is often referred to as the dominant theory of learning today[6]. Constructivists view the learner as a *'active entity'* and hold that knowledge is constructed in the learner's mind rather than transmitted from teacher to learner[7]. Central to constructivist theory is the concept of the '*mental*

model' and that these mental models are expanded by the assimilation of new information received through our experiences[8].

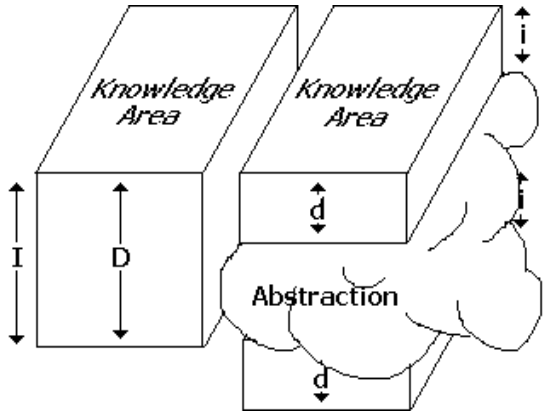

Figure 2 : Abstraction of Data

All of us have mental models of the things around us. New experiences challenge those models and, in response to those challenges, we modify and expand our models. This continuous construction and re-construction of mental models is the constructivist process of learning. Whether or not our current model is theoretically perfect is largely irrelevant. What matters is that, in our current environment and therefore subject to our current experiences, the model is adequate. It is all a matter of perception.

Those of us who learned computing in a UNIX (or DOS) environment often regard the youth of today, with totally GUI experience, as being disadvantaged. "Those poor kids," we say, "don't know how it works." What we must bear in mind is that, while the student's mental model of a computer might be somewhat primitive from our point of view (*"an electronic brain"* or merely *"a black box")*, the model is effective — it works. As Ben-Ari points out that even after a full semester of Pascal, students' knowledge of the conceptual machine underlying Pascal can be very fuzzy but arguing about *'alternative frameworks'* of language syntax or semantics might only be a cause of psychological grief for the students[6]. After all, this is the basis of the Object-Oriented Paradigm. Encapsulation and Information Hiding lie at the heart of OO. Once an object has been created and has methods, what does it matter how the methods perform their functions? What does matter is that they perform their functions. It is the concept of what the object is supposed to do — the '*mental model'* of the object — which makes that object useable. This is abstraction, and this is why the authors hold that an SE can operate on a reduced depth of understanding of the intricacies of CS. As long as the SE can understand that a VLSI chip can handle data at a specified rate and perform a specified range of functions, the knowledge of electrons following golden paths across a silicon chip is of little or no consequence. Hence we can reduce the depth of the volume of knowledge simply by abstracting away the detail. At Edith Cowan University (ECU), a new (1998) syllabus does just that.

2. A New Computer Technology Modelling Abstraction

The ACM/IEEE Computing Curriculum 1991 defines international benchmark standards in computer architecture education as a pre-requisite chain of seven units. However, computer design and manufacture has changed rapidly in the last decade. Assembly Level Manufacturing, Application Specific Integrated Circuits and Surface Mounted Technology have all led to an ever-decreasing unit price and a resultant low cost PC with a standard architecture and modular

construction. The relevance of the standard computer architecture is therefore in question and yet according to the 1991 ACM/IEEE-CS report, curriculum planning should be driven by the outcomes expected for students[9]. In 1996 an ECU market audit led to a set of guidelines developed for the type of skills expected of computer science graduates entering the field of computer and network support[10]. Using the criteria developed a random selection of ten, final year Edith Cowan University (ECU) computer science undergraduates were interviewed from a graduating population of approximately one hundred. The computer science degree at ECU is level one accredited, the highest, by the Australian Computer Society yet none of the students interviewed had the skills expected by prospective employers. Interviews with graduates employed in the field indicated that they were largely self-taught. The lack of relevance of much of the technical detail in the standard computer technology curriculum appeared to be the single most important factor for this lack of technical knowledge and the authors hold that an effective solution to this *'detail overload'* is abstraction. According to Clements within the field of computer technology education academics must continually examine and update the curriculum, raising the level of abstraction[4] which is in keeping with the ACM/IEEE Computing Curriculum 1991 in which abstraction is a recurring concept fundamental to computer science[9].

Models are used as a means of providing abstraction and hence aids to communication and controlling detail. Diagrammatic models should have the qualities of being complete, clear and consistent. Consistency is ensured by the use of formal rules and clarity by the use of only a few abstract symbols. Levelling, in which complex systems can be progressively decomposed, provides completeness. According to Cooling[11], there are two main types of diagram: high level and low level. High level diagrams are task oriented and show the overall system structure with its major sub-units. Such diagrams describe the overall function of the design and interactions between both the sub-systems and the environment. The main emphasis is 'what does the system do' and the resultant design is therefore task oriented. According to Cooling, 'Good high-level diagrams are simple and clear, bringing out the essential major features of a system'. By contrast, low-level diagrams are solution oriented and must be able to handle considerable detail. The main emphasis is 'how does the system work'. However, models should have the following characteristics: diagrammatic, self-documenting, easy to use, control detail and allow hierarchical top down decomposition. Computer technology can be modelled using symbolic Boolean algebra (NOR, NAND gates) without the complexity of their implementation in different switching technologies eg TTL, CMOS, BICOMS etc. The underlying switching technology is not relevant at this higher level of abstraction. These gates may be implemented using solid-state electronic switches or even gas state electronics i.e. thermionic valves. At this lower level, the basic implementations of solid state switching may be described by models directly relevant to engineers at this level of operation. Similarly details of semiconductor switching may be modelled using abstractions independent of the underlying details of quantum mechanics. Logic gates may be connected to create combinatorial and sequential circuits and hence functional units such as Read Only Memory (ROM) etc. Such functional units may also be modelled but using a higher level of abstraction. At an even higher level of abstraction computer technology can be modelled as a collection of programmable registers. Computer technology can therefore be described using a progressive range of models based on different levels of detail e.g. semiconductors, transistors, digital circuits.

Hardware Description Languages offer precise modelling notation, however such notation may not be suitable for a first year course in computer technology. Based on this modelling technique a new pedagogical framework for teaching computer and network technology was designed, implemented and evaluated. Using this modelling technique each device within a PC (microprocessor, hard disc drive etc) is treated as a node that is a data source/sink capable of, to various degrees, data storage, processing and transmission. Results have shown that even though technical detail is lost, this model is conceptually simple, controls detail by abstraction and may allow students to easily make viable constructs of knowledge based on their own experience. Work to date indicates that modelling the PC as such a collection of nodes provides a good constructivist framework allowing technical detail to be introduced in a controlled, top-down manner that is readily understandable to students from all disciplines. The nodal model provides abstraction and hence is independent of architectural detail and can therefore accommodate rapid changes in technology. Significantly the lower levels of technical detail such as digital techniques are not taught.

3. Benchmarks and bandwidth

Consumer magazines use benchmark suites to evaluate PCs and publish their results to provide a basis for comparison. Results, however, can merely be confusing. Consider Table 1. In some tests one machine performs better than the other and in other cases the situation is reversed. Consider the range of values displayed. These inconsistent and ambiguous results raise a lot of questions. Some examples are: (1) What difference in performance can a user expect if the benchmark value result is higher by one or two units — or a factor of ten? (2) What difference in performance can a user expect between a Pentium 100 and a Pentium 200? (3) As a user, how is the difference in performance manifested and perceived? (4)How do you compare the performance of a hard disc drive to that of a microprocessor?

Table 1 : PC Benchmark Suite

PC Benchmark Suite		
Benchmark	**Gateway G6 300**	**IBM Aptiva EQ3**
Business Winstone 98	20.63	18.33
CD-ROM WinMark 98: Overall	1,556.67	1,350
CPUMark 32	772	550.33
Business Disk WinMark 98	1,380	939
High-End Disk WinMark 98	3,783.33	2,736.67
Business Graphics WinMark 98	93.13	105.67
High-End Graphics WinMark 98	146	130

This is the sort of confusion which faced a team from ECU teaching a course in hardware maintenance. They concluded that benchmark suites did not provide a useful basis for a conceptual model of a PC for the students at a lower undergraduate level who form the majority in their classes. To be of practical value for undergraduates — or PC users in general — a measurement standard must be relevant to human dimensions or perceptions (and, preferably, use reasonably sized decimal units). Bandwidth was the unit selected.

4. Making Bandwidth User-Friendly

Many of the students in the class come from outside mainstream CS, some from Multimedia,

some from Education courses, some from the Business school, some from other Universities (this remains the only unit of its type available in Australia). For most of these people, bits, Bytes and Megabytes are just as abstract and vague as Whetstones or Dhrystones so the team looked for a common unit in which to express bandwidth. They realised that PC users work in a GUI environment, and that a major concern was the response time of the machine — how fast the screen would change — so the team decided to measure the performance of a machine, or part of a machine, by the time it took to transfer the amount of data required to support a full-screen image. Frames per second was then the chosen unit, being a model readily acceptable to people from outside of the computing mainstream but one that is, later in a CS course, readily transmutable back into Megabytes.

Table 2 : Bandwidth

Bandwidth			
Device	**Clock Speed (MHz)**	**Data Width (Bytes)**	**Bandwidth (MB/s) B=CxD**
Processor	400	8	3200
DRAM	16	8	128
Hard Disc	60rp	90KB	5.2
CROM			4.6
ISA Bus	8	2	16

5. A New Benchmark

Each hardware node (microprocessor, hard disc drive etc) could now be treated as a quantifiable data source/sink — effectively a *'black box'* — measured in Frames or MB, with an associated data transfer characteristic (Frames/s or MB/s). (The nodes are now referred to as B-nodes (Bandwidth-nodes).) This approach allows the performance of every node and data path to be assessed by a simple, common measurement — bandwidth — where

Bandwidth = Clock Speed x Data Path Width

with the common units of Frames/s (MB/s). (ref: Table 2)

6. Testing the New Unit

A 'C' program was used to transfer data between two nodes, a Hard Disc Drive (HDD) and Synchronous Dynamic RAM (SDRAM), flagging the start and stop of this operation on the parallel port. An oscilloscope (100 micro second resolution), connected to this port measured the data transfer rate in MB/s. The team were able to detect the influence of HDD caching and also track and cylinder latency thus verifying the experimental method.

To a first approximation, smooth caricature animation requires approximately 5 frames/s (5.85MB/s). (Obviously sub multiples of this unit are possible such as quarter screen images and reduced colour palette such as 1 byte per pixel.) A single, uncompressed 640x480, 4bytes/pixel-video image was generated and transferred from the HDD to both SDRAM and a third node, the video adapter card. The data transfer rate from HDD to SDRAM was 1.48MB/s, which can be expressed as 1.21 frames/s. From the HDD to video adapter card the data rate was 1.37MB/s i.e. 1.1 frames/s. The data transfer rate for the video card is specified as 18.6MB/s i.e. 15.1 frames/s. Obviously, the limiting factor here is the HDD which is unable to provide bandwidth for smooth motion in an animation sequence.

Nodes typically operate sub-optimally due to their operational limitations and also the interaction between other slower nodes. For example, a microprocessor may need two or more clock cycles to execute an instruction. Similarly a data bus may need multiple clock cycles to transfer a single data word. The simple bandwidth equation can be modified to take this into account:

Bandwidth = Clock x Data Path Width x Efficiency.

The Intel 8088/86 required a memory cycle time of 4 clocks cycles (Efficiency = ¼) however, for the Intel 80x86 series, including the Pentium, the memory cycle time consists of only 2 clocks (Efficiency = ½) for external DRAM A 100Mhz Pentium has, therefore, a bandwidth of 400MB/s.

$$B = C \times D \times E$$

$$B = 100(MHz) \times 8 \text{ (bytes)} \times 0.5 = 400$$

Other devices can be modelled in a similar manner.

B-nodes also allow recursive decomposition. Hence a PC can be described as a B-node or a collection of devices all modelled as B-nodes. Each device can also be modelled as a collection of B-nodes. By example a hard disc drive, itself a B-node, can be decomposed into B-nodes that represent the electromechanical devices (motors, CHS architecture) and the hard disc controller (ENDEC, ECC, etc).

7. In-Class Application

In 1998 the B-node model was used in the classroom for the first time. Using this conceptual framework (*"mental model"*) the PC is considered as a series of nodes that can store, process and transfer data. Student understanding was evaluated by means of two assignments in which they were required to obtain the technical specifications for a PC and construct a nodal model. (Prior to the introduction of the B-node model most students were unable to predict the likely performance of a PC and identify nodes (devices) that would significantly handicap performance — student assignments resulted almost exclusively in a list of hardware details copied directly from technical literature with little or no critical analysis.)

One student wrote: "The lack of meaningful and comparable technical specifications makes the task of calculating the performance of a PC and it's individual components a difficult one. The computer industry appears to be a law unto itself, with incomplete or non-existent technical specifications. The use of standards, terms and abbreviations are not comparable across different systems or manufacturers. This all leads to frustration and confusion from consumers and users of these computer systems and components." Further, when given technical specifications for a PC: "Sounds very impressive, yet by undertaking the exercise of converting to the components common units the relative performance of the PC and it's individual components can be measured and conclusions drawn. You will finally be able to see exactly what you are purchasing, its strengths, weaknesses and overall value."

8. Conclusions

Abstraction provides an effective means of reducing the volume of information which has to be transmitted in the classroom while allowing the students to formulate an adequate conceptual

model of the content of the knowledge area. B-Nodes present each device within a PC (microprocessor, hard disc drive etc.) as a data source/sink capable, to various degrees, of data storage, processing and transmission. Independent of architectural detail, the model can, therefore, accommodate rapid changes in technology. Lower level of technical detail such as digital techniques need not be taught in the initial stages of a degree yet this detail is controlled by the abstraction and may be introduced — expanding the student's *'mental model'* — later in the course.

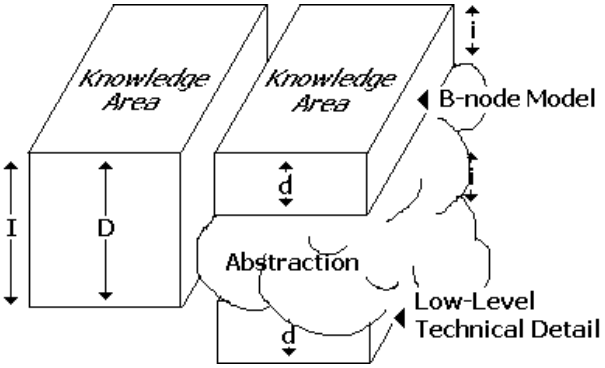

Figure 3 : The B-Node Abstraction

Benchmarking by bandwidth has been shown to be an effective means of cutting through the confusion of benchmark suites as displayed in Table 1. It is simple, readily measured and, above all, readily understood. Abstraction of megabytes into frames per second has proven an effective means of delivering the concept of relative hardware performance even to people from outside of the computing mainstream while preserving the integrity of the underlying technical detail. An independent review of the Computer Installation and Maintenance (CIM) unit (utilising these abstractions as part of the pedagogical technique) conducted by Dr Tony Fetherston of the Multi-Media and Learning Technologies department at ECU found that: "80% *[of students]* would recommend this unit; 75% found the practical sessions useful; 70% found the unit relevant to their needs and 55% think this should be a compulsory unit."[12] Another metric of the success of the hardware units is that of student demand for those units. "The response from students was overwhelming. The initial quota of 100 students for CIM was exceeded with 118 students enrolling… The student attrition rate was 8.5% with a subsequent unit failure rate of less than 10%."[10]

Abstraction, therefore, has, in the view of the authors, proven to be an effective tool for reducing the quantity of detail which needs to be transmitted in the classroom. We believe that its application in other subjects holds the key to the creation of SE curricula which cover the necessary CS areas of knowledge while preserving time and space for engineering and the physical sciences.

Bibliography

1. ACM Curriculum Committee on Computer Science, Curriculum 68: Recommendations for Academic Programs in Computer Science *Communications of the ACM*, 11, (3)pp. 151-169, Mar, 1968.

2. E. Roberts, R. Shackelford, R. LeBlanc, and P.J. Denning, *Curriculum 2001: Interim Report from the ACM/IEEE-CS Task Force* Proceedings of ACM/SIGCSE '99 pp. 343-344, 1999. ACM. New York NY(USA).http://www.acm.org/pubs/articles/proceedings/cse/299649/p343-roberts/p343-roberts.pdf (August 10, 1999).
3. IEAust Working Group on Software Engineering, Software Engineering as a Professional Engineering Discipline: Discussion paper Mar, 1999. (unpublished).
4. A. Clements, Computer Architecture Education *IEEE Micro*, pp. 10-12, May, 2000-Jun 30, 2000.
5. G.A. Ford and N.E. Gibbs, A Master of Software Engineering Curriculum: Recommendations from the Software Engineering Institute *IEEE Computer*, pp. 59-71, Sep, 1989.
6. M. Ben-Ari, *Constructivism in Computer Science* Proceedings of SIGCSE'98 pp. 257-261, 1998. ACM. New York, NY (USA).
7. D. Aharoni, *Cogito, Ergo Sum! Cognitive processes of students dealing with data structures* Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education pp. 26-30, 2000. ACM. New York, NY (USA).
8. D.S. Brandt, Constructivism: Teaching for understanding of the Internet *Communications of the ACM*, 40, (10)pp. 112-117, Oct, 1997.
9. IEEE-CS/ACM Joint Curriculum Task Force. (91) Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force [Web Page]. http://www.computer.org/education/cc1991/eab1.htm [June 1, 2000].
10. S.P. Maj, G. Robbins, D. Shaw, and K.W. Duley, Computer and Network Installation, Maintenance and Management - A Proposed New Curriculum for Undergraduates and Postgraduates *The Australian Computer Journal*, 30, (3)pp. 111-119, Aug, 1998.
11. J.E. Cooling. *Software Design for Real-time Systems*, London, UK: Chapman and Hall, 1991.
12. S.P. Maj, T. Fetherston, P. Charlesworth, and G. Robbins, *Computer & Network Infrastructure Design, Installation, Maintenance and Management - a proposed new competency based curriculum, based on international best practices, fully articulated to non-university education programs.* Proceedings of the 3rd Australasian Conference on Computer Science Education 1998. ACM. New York, NY (USA).

Biographical Information

RICK DULEY graduated with First Class Honours in Computer Science in 1996. He is currently a Doctoral Research Student at Edith Cowan University working in the field of Software Engineering Education. Coming from a Heavy Industry and Mining background he has a special interest in things industrial and in the application of engineering principles to software construction. His interest in abstraction in the classroom arises from his experiences in teaching SE principles in programming classes. r.duley@cowan.edu.au

Dr S P MAJ is a recognized authority in the field of industrial and scientific information systems integration and management. He is the author of a text book, '*The Use of Computers in Laboratory Automation*', which was commissioned by the Royal Society of Chemistry (UK). His first book, '*Language Independent Design Methodology - an introduction*' , was commissioned by the National Computing Centre (NCC). Dr S P Maj has organized, chaired and been invited to speak at many international conferences at the highest level. He has also served on many national and international committees and was on the editorial board of two international journals concerned with the advancement of science and technology. As Deputy Chairman and Treasurer of the *Institute of Instrumentation and Control Australia (IICA*) educational sub-committee he was responsible for successfully designing, in less than two years a new, practical degree in *Instrumentation and Control* to meet the needs of the process industries. This is the first degree of its kind in Australia with the first intake in 1996. It should be recognized that this was a major industry driven initiative. p.maj@cowan.edu.au

ALASTAIR BOYANICH is a Computer Science Graduate of 2000 currently studying a Masters by Research at Edith Cowan University in the field of system performance. His interests are in machine performance and distributed processing with programming experience in private industry. iso9660@yahoo.com