Bootstrapping Computer Science in Old North Wales

K.A.Hawick and H.A.James

Computer Science Division, School of Informatics
University of Wales, Bangor
Dean Street, Bangor, LL57 1UT, UK.

{hawick,heath}@bangor.ac.uk

Abstract

We describe our experiences in setting up a new Computer Science department in an established University within the UK. We drew upon our experiences in teaching Computing in Australia, the USA and the UK. We discuss cultural differences and constraints upon us in designing and teaching a BSc in Computer Science. We also relate teaching and culture bootstrap effects from having to start a whole new department from origins in an Engineering and Mathematics School. We relate our design decisions and experiences in setting up teaching and learning methods and appropriate assessment practices for the rapidly broadening discipline of Computer Science. We also describe our PhD programme and our early experiences of a taught MSc programme.

Keywords: curriculum design; teaching and learning methodology; assessment techniques; computing culture

1 Introduction

Computer Science is perhaps one of the most popular subjects amongst students worldwide at present and is certainly one of the fastest growing and broadening disciplines. Through a curious set of circumstances our University did not have a Computer Science department and we were given the fascinating opportunity of setting one up.

Establishing any new department it not an easy task but the competition for staff makes establishing a Computer Science department particularly challenging. Our new department is part of a School that contains the Mathematics and the Engineering departments so we had some material and some excellent colleagues to draw upon. In particular some material for Software Engineering was available from existing degrees as was a portfolio of excellent mathematical modules that could be used as options. Nevertheless a great deal had to be done.

In this paper we focus on our flagship undergraduate degree programme – our BSc in Computer Science and describe how it was designed and how it has grown. It was with great satisfaction that we saw a cohort of students achieve 1st, 2nd and 3rd class honours degrees from this programme in July 2002, and we were also delighted that the British Computer Society (BCS) accredited (British Computer Society 2001) the degree through to 2005. Perhaps most satisfying has been the steady growth and nearly annual doubling of student enrolment to the degree. As well as describing our curriculum design we relate our design decisions and thoughts on teaching and learning methods and assessment for our new degrees.

We also discuss some of the other departmental programmes we have inaugurated and in particular how they interrelate with the main undergraduate teaching programme in Computer Science. We describe offshoot degrees in interdisciplinary areas and developments in MSc degrees. Our PhD programme has absorbed many of our School's students and working with those students has given us some important insights into what was missing prior to the Computer Science programme. Finally we describe some of attempts at filling the gaps between formal degree programmes and employment for graduates. We have instigated a technology transfer programme and a set of professional short courses that provide practical experience on specific software packages.

2 Background

The University of Wales is organised as a federation of different geographical units, which were formerly known as colleges. The University of Wales, Bangor located in North Wales has been established since 1884 but only recently made the decision to establish a computer science programme. Despite the obvious popularity of computing and Information Technology (IT) in the last 20 years, the federal University had only allowed computer science programmes at the other campus sites. A decision was made in 1999 to initiate a brand new Computer Science programme and we were given the opportunity to establish this. Some software engineering teaching modules had been established as part of an engineering programme but otherwise we have had to develop material from scratch. This is a somewhat unusual set of circumstances for a UK University.

The University of Wales, Bangor is located in the North Wales coastal town of Bangor and has around 8000 students in total. The School of Informatics was formed as an amalgamation of Engineering, Mathematics and the new Computing Division as a way of addressing resourcing issues connected with dropping student numbers. Happily, computing student intake has nearly doubled each year since the inception of our Computer Science programme and we now have an annual cohort size of over 60 computing undergraduate students.

3 BSc in Computer Science

The Association for Computing Machinery (ACM) provides an excellent framework (Association for Computing Machinery 2001) for considering a Computing Curriculum and we used this as a starting point for design of our degree. There has been a lot of discussion in the literature about what balance of content can be realistically achieved for a modern computing degree (Piner 2001). We realised that it would be important to design a degree that had a core programme that was acceptable internationally and not just in the UK. The English/Welsh system of 3 year honours degrees is somewhat constraining (Quality Assurance Agency for Higher Education 2000) and this led us later to set up an MSc degree that would help provide that "extra year" that seems to us to be a valuable aspect of the Scottish and Australian honours degree systems.

Another recent trend is the pressure on Universities in the UK to widen their entry routes and in particular to lower their Mathematics admissions criteria (Mosley 2000, Maskell & Robinson 2001). We felt this could be done providing we took on the burden of teaching students the necessary mathematics for computer science within the degree programme itself. This added even further to pressure on curriculum space.

A final constraint on any new programme is to accept the broadening of Computer Science as a discipline and the need to cover new specialist topics within a limited curriculum space and with limited staffing resources.

Accepting the constraints of our system we adopted the degree philosophy of:

- Providing a core programme of discrete mathematics and programming and algorithms in the first year.
- Providing a fairly broad exposure to concepts and other specialisms in both first and second year
- Providing modules in all years that expose the students to as much breadth as possible but provide customisable strands of specialism.
- Ensuring the core programme is workable as fast as possible and make use of staff / research group special interests in setting the possible options.

The UK system has adopted a modular system for organising degree programmes. In our University degrees consist of 12 modules per year all of (usually) the same credit weighting. We have found this system works well for the more advanced years and specialist subjects. It is not always easy to work around it for first year subjects that require a lot of foundation teaching that does not easily fit into a single module. Generally we have circumvented this problem by establishing what are effectively "double modules" one following on from the other in our semester system. The modular system has proved useful however in designing offshoot degrees from our main Computer Science degree. We have been able to reuse some modules between degrees. We are also finding this feasible for a portfolio of MSc modules.

3.1 Curriculum Design

We have described the constraints upon us and now describe our curriculum design. We were able to make use of some common modules from mathematics and engineering degrees but have had to design many modules from scratch. Before giving details on individual modules we first lay out the strands or themes that we wished to cover.

3.1.1 Algorithms

We felt that algorithms and the necessary mathematics had to have a very strong place in our programme and to that end we chose a strand of eight modules to embody this. Two custom designed discrete maths modules were designed for first year and an additional module in applied algorithm design is used in first year. A module on probability and statistics was adapted from pure maths modules to provide concepts exposure module in first year. In second year custom modules on complexity theory and automata theory are compulsory and in third year modules on graphical algorithms and logic are compulsory. We make available a set of continuous (calculus and numerical analysis) modules.

In the early years of the degree there was a feeling that we had set the demands of these modules too high with students finding them quite difficult. However as the cohorts have increased in size we now see student marks following a more normal spread and achieving quite satisfactory averages.

3.2 Programming

It is always controversial which core programming language to adopt or even whether one should be adopted. We decided that for all the usual pragmatic reasons we would try to use Java as our first teaching language. Following the ACM curricula advice however we were also keen to ensure students would be exposed to other programming paradigms. We chose Scheme to use as our first year AI language and to use Haskell as a functional programming language to introduce in third year. A controversial decision was whether to expose students to assembly language. Our engineering degree made use of 68000 series assembler language in teaching simple control systems. We decided that a better compromise would be to introduce a modern but lightweight control chip such as the PIC series and to teach students how to program this in C but still expose them to some low level machine/assembly level code. The engineering degree modules have embraced this compromise too and we see some measure of success in that final year and postgraduate projects are being successfully carried out using the PIC system.

We have had difficulty explaining to engineers the importance of teaching programming concepts as well as mastery of just one language. We have also had the criticism from our industry collaborators that they want to employ graduates who can program a particular language and can use a particular integrated development package. We are trying to resist such compromises and continue to teach programming concepts. We have however adopted

the supplementary practice of offering short intensive professional courses in particular programming languages and packages. These are run over the summer.

3.3 Software Engineering

It is notoriously difficult to teach realistic software engineering practices to undergraduate students. Our School has an active research group working on some of the more recent software engineering methodologies and practices. This group is closely involved in running what we call our "Software Hut" modules. (The word "Hut" being a diminutive version of "House"). In addition to modules covering software engineering ideas and methods in first and third year, the "Hut" modules involve a group project carried out in second year. Groups typically involve four or five students. A recent idea has been to have groups design a software system in first semester and then swap designs and have them build and test some other group's design in second semester. Students find this challenging but it does seem to impart some of the key ideas and frustrations of working in a realistic software team environment.

3.4 Distributed and Network Computing

Our largest research group works in Distributed Computing and we were keen to ensure good coverage of relevant distributed computing ideas in the curriculum. We cover networks and communications in both second and third years and also expose the students to various distributed programming models in modules covering operating systems and parallel systems. Many students are attracted by this area and see themselves as working in the web services; telecommunications; systems administration or e-commerce areas. We have established good relations with several large companies such as Vodafone and BT and are able to find summer and work placements for our undergraduates and graduates in this area. There is also a large uptake of final year projects in this subject. Relatively recent national interest in grid computing has built on this area and we are introducing a new MSc on Distributed Computing Systems and Computational Grids.

3.4.1 Artificial Intelligence

According to some practitioners of AI this field is just emerging from a "winter" of disinterest perhaps brought on by over-hyped expectations in the late 80's and early 90's. We believe it is an important area that is growing in applicability again. We wanted to provide an introductory core set of modules and a set of core specialist options for second and third year. A new concept we have introduced is that of a specialist module on agent technologies. This has proved a very popular option with final year students.

There was a long-standing interest in applied neural network techniques in the engineering department and one of our own research interests is in smart and mobile systems. It was therefore natural to set up this specialist strand of teaching modules. We have also set up a distributed robotics research activity that supports several undergraduate projects.

3.4.2 Parallel Computing

In some ways parallel computing is much more mainstream than it was ten years ago and not such an esoteric part of the curriculum. We cover parallel computing and high performance computing in a specialist module in third year. We also introduced material on concurrency in second year. We found it was feasible to build on student interest in Java programming from first year and use the Java threads package to allow students to experiment with simple concurrency ideas. We also link concurrency and parallel programming ideas to practical exercises using multi processor systems and computer clusters. Students seem attracted by this area and it has led to several final year projects.

3.5 Databases and Systems

Databases and Information systems play an important part in student's appreciation of IT. We introduce database concepts in first year and build on these ideas for another compulsory module in third year. Students seem to recognise that this is another area of employability and the modules seem popular.

3.6 Theoretical Computer Science

In addition to the modules we described under the algorithms strand, we adopted many of the modules available under our Mathematic degree as possible options for Computer Science. These include operational research; computational geometry; Markov chains and statistical pattern recognition. We are presently considering how far we can take the theoretical computing strand within the undergraduate curriculum and are preparing an advanced module on the theory of languages and machines. Clearly some students do enjoy theoretical computer science and computational science.

3.7 Miscellaneous Computing Modules

There are several other miscellaneous computing modules covering other important aspects of computer science and programming. There are various topics that we were under pressure to include in the curriculum that while important is not, we felt, justify having a whole module to themselves. We have managed to combine topics like computer graphics and Human Computer Interaction into a single module for third year for example. Other areas like professional ethics and transferable skills we felt are very important but can be taught in context of other modules rather than having a dedicated module. This is an area where the modular system is less than ideal, and we face curriculum design issues and compromises that would not arise in a course-based curriculum. When we designed the Computer Science programme we tried to cover the key areas that have been recognized by international computing bodies like the Association for Computing Machinery (ACM). We believe we have succeeded and that our curriculum coverage is not just to British standards but is in fact to an International level.

4 Teaching and Learning

We are employing a fair variety of teaching and learning methods in our modules. These are essentially designed to ensure students are exposed to key concepts and ideas in computer science, to reinforce the key ideas, and also to encourage students to think laterally around a subject. In this section we briefly describe these methods. We have had to change the existing "traditional engineering" approach to teaching and learning and essentially to bootstrap a new computer science culture. This has been an interesting and perhaps unusual experience.

Traditionally, the main method of information delivery in the University setting has been lectures. The largest lecture class sizes are found in those modules that are common to all the School's degree programmes, such as first-year programming. We are fortunate in that the class sizes for most lectures in the second and third years of our degree programmes are fairly small. This allows us to run our classes in a more interactive fashion than would be possible for, say the first-year classes. We are encouraging the more "traditional" lecturers to adopt electronic presentation programmes to allow demonstrations of concepts during lecture times. Using these programmes has the additional benefit of allowing lecturers to trivially make their lecture slides available to the class after the lecture. Like most institutions we are having a challenging time encouraging the students to take lecture notes when they have the knowledge that the lecture slides will be available electronically at a later stage. We attempt to counter this problem by stressing that the lecture slides are incomplete in terms of the module's examinable information.

We also introduced small-group tutorials to reinforce ideas presented in lectures and also to encourage students to think laterally. Similar to other institutions we find that some students take a substantial amount of time to become comfortable with some concepts and also to see how to generalise a sometimes-specific example to the general case. We distribute the tutorial questions to the students the week before their tutorial; at the tutorial they are expected to participate in the discussions in order to satisfy the continuous assessment requirements of their course.

One of the main methods that we use for teaching computer science is that of supervised and unsupervised laboratory sessions. We are fortunate that we have a number of well-equipped laboratories featuring dual-boot Unix and Windows PCs. Unfortunately the computing culture of the building was such that only the systems administrators used Unix, and most of the staff (and hence the students) used Windows. This problem was addressed by stressing the importance of not letting the waning "Unix culture" in the department die; we immediately required that all our continuous assessment work be carried out using the Unix operating system. This has had a marked effect in the perceived low-level computing knowledge of most students. We employ post-graduate students to supervise laboratory sessions in which students are given exercises to complete and problems to solve which draw on knowledge and concepts from lectures and tutorials. We are also attempting to set up a "guru on duty" system where outside scheduled laboratory hours there will be a staff-member (or post-graduate assistant) available in a well-known location, willing to provide help on any computer science module.

We also encourage students to spend as much time as possible thinking about the material presented in lectures, and to investigate any of the topics that we cover in lectures more fully in their own time. There are two avenues where we see computer science students actively doing these things: firstly by 'playing' on the computers; secondly, by doing independent reading. Until we arrived we feel that the staff did not properly understand the qualities that make a competent and confident programmer; students were not encouraged to 'play' on the computers. We suspect this came from the fact that most staff were engineers and in this discipline students had to be supervised by a laboratory technician at all times. Independent reading was a vital component of our own undergraduate educations and we encourage students to read certain recommended textbooks and also discuss useful programs that make up the "Unix culture."

Associated with the above point is the reinforcement of good study and research habits. It is a fact that there is a huge body of material and experiential knowledge available on the Internet via the World Wide Web. Tutors and laboratory demonstrators are often encouraged to help students, not by giving them direct answers to their questions, but by telling them how, or sometimes where to search for the answers. As would be expected, we have found that students treated in this manner show a more in-depth appreciation of the concepts covered in the computer science curriculum.

In the second year we run a small group project. Part of the software engineering module, this project allows students to experience the difficulties in writing software and specifications for larger groups, and also introduces them to the more social aspects of software engineering in a team environment. We feel this aspect of our curriculum is vitally important, as traditionally computer science students are not by nature very extroverted people, so benefit immensely from the experience. We also emphasise the importance of rigorous (unambiguous) software specification by making groups exchange specification documents before starting the implementation.

Possibly the most important form of assessment in our three-year computer science with honours programme is the individual project. We have found that most of our students look forward to this module. The individual project allows the student to take ownership of a quite substantial fraction of their assessment (three modules' worth in their final year) that often helps to raise their module average. The project is typically agreed upon by the student and an academic staff member, who both negotiate learning outcomes and project milestones. Usually the student project topic is affiliated in some way with the academic's research group, or is at least in line with the academic's research interests. Due to a lack of understanding of what computer science really is, we had to work around some misconceptions by the "traditional"

staff that computing projects could be of 'information systems' and IT project quality. Some computing topics in recent years have included: building programmable robots using Lego; building distributed web and network applications; experimenting with virtual reality and advanced graphics systems; building programmable remote control systems; building wireless access protocol mobile phone applications; programming personal digital assistants to interact with wireless networks.

5 Assessment Methods

In this section we discuss some of the assessment methods that we are using in our computer science degree. When considering the assessment for a module, there is a balance to be struck between the continual assessment and end-of-semester unseen examination. It is a fact that some students are more disposed towards the bulk of assessment being continual with only minimal contribution from the examination; whereas some students would prefer the bulk of the assessment contribution from the examination.

Unseen examinations are the traditional method which universities assess subjects. We are, however, finding that more and more computer science material must be examined by practical demonstration of understanding due their less theoretical natures. It is for this reason that we are migrating from the situation in which nearly every module has an 80% weighting on the exam, to many lowering this to 50-75%. When we arrived it was also 'custom' for exams to be 1½ hours in duration; we encountered a certain reluctance to suggestions that exams be made longer in order to examine more of the semester's taught material. Traditionally the exams had been structured in the form that required the students to answer all the questions of approximately a third of the paper, and then for the remainder choose one of, say five, questions. We found that this format allowed students to know only one section of the paper really well, which exhibiting only basic knowledge of the rest of the course, which the students were capitalising on. We also found reluctance to change the format of exams to an all-compulsory format.

Supervised laboratories attached to some modules also have the requirement that logs be kept of the experiments performed or the exercises attempted. Typically these laboratories are of the more physical variety; students are expected to submit word-processed reports showing graphs, tables, or experimental data.

Most modules have some programming practical exercises or essays to hand in. This is the traditional form of continual assessment in computer science, and is often the way in which undergraduates learn to be competent programmers; it is by doing assignments they get the practice needed to become familiar with the programming concepts they need. This also allows academic staff to provide much-needed feedback to the student and also allows them to detect any weaknesses in the computing curriculum. We have resisted the temptation to assign more practical programming assignments to the students as many already feel they are being over-examined and do not have time to absorb the course material at their own rate. We are also tempered by the need to ensure that any material presented for continual assessment does indeed originate with the student submitting it; with the dearth of information and assignments available on the Internet, plagiarism is on the increase, and we are having to investigate mechanisms to detect it in the face of ever-increasing student numbers.

When we arrived the mechanism that students had to use to submit assessed work was to individually place it in an open pigeon-holed box unit. Unfortunately these boxes were in an area that was usually under the supervision of a clerical assistant; there were times when the assistant was busy elsewhere or has to leave early. At these times, the security of the boxes could not be ensured. Clearly this situation was not acceptable. Another solution was to have students email the electronic version of their exercise to the marker; this met with considerable resistance as most of us get too much email already, and this was not easy to do without creating a new email account for every module. We eventually had the systems administrators create a web-based hand-in program through which the students could submit their assignment (in one of a list of acceptable formats) either from within the department or remotely. Lecturers (or teaching assistants) could use this programme to download and mark the assignments. Finally, the same program could be used to return the assignments to the students with feedback. A useful by-product of requiring the students to use this program was that their assignments could be fed into automated plagiarism detection software.

Often associated with group projects are oral presentations. Students are required to make short reports describing such things as the problem, their approach to the problem, and a discussion of any other relevant details. In addition we reserve the right to use oral examinations to resolve situations of suspected plagiarism.

The final method of assessment is the project dissertation. Coupled with the individual project, the dissertation is the main deliverable. The dissertation is a substantial report and often contributes to the body of knowledge in an academic's research group. The processes involved in preparing and writing the dissertation educate the student in the scientific process and the style that is expected when writing scientific documents.

6 Offshoot Degrees

Once our BSc in Computer Science and its curriculum and module set was established it was attractive to consider how a greater set of programme options and degrees could be offered to students. Our experience has been that many students want to study computer science but also want to study an associated applied discipline. While we have we believe been successful in offering an attractive set of options within the degree, we also wanted to offer combined programmes. At a time when other applied sciences and other departments were seeing dropping numbers of student enrolments, they were also

pleased to collaborate over joint programmes. Figure 1 shows the joint degrees we now have on offer.

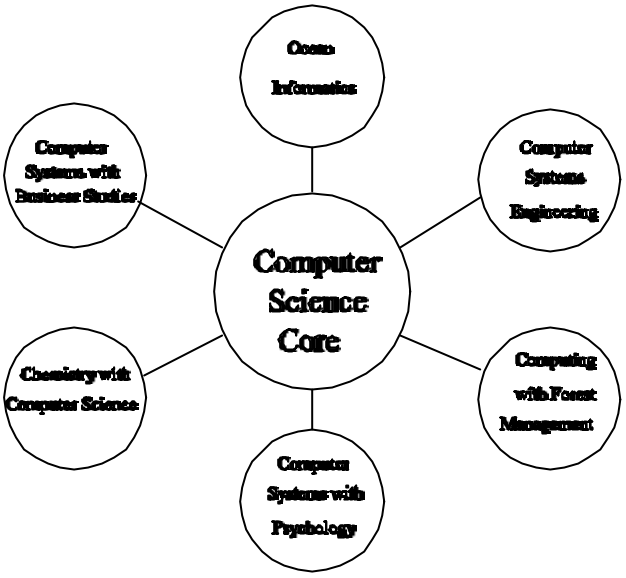

Figure 1 Offshoot degrees from our BSc in Computer Science

Computer Systems Engineering emphasises engineering aspects of computing including software engineering and electronics. Other science departments such as chemistry and ocean sciences had applied modular programs that it was fairly easy to combine with our computing modules. These joint degrees have smaller student cohorts but seem to be working well so far.

7 MSc in Distributed Computing

Not content with the undergraduate portfolio of degrees we set out to establish a portfolio of taught MSc degrees. This endeavour was motivated by a need to provide a fill-in year to bridge the gap between our three year honours degrees and PhD study. Drawing on our own research interests in distributed computing, we designed an MSc in Distributed Computing and Computational Grids; our only pre-requisite of applicants was they have a recent numerate degree. The general ethos for this MSc is essentially how to be a scientist. We have subjects with a range of sophistication, from how to perform a literature review, to advanced distributed computing concepts. As in our undergraduate programmes, we aim to re-establish the 'Unix' culture through a broad education of graduates in the different computer operating systems available today.

8 PhD Programme

When we arrived at our University we were somewhat surprised to discover that there were very few postgraduates in even the general area of computer engineering. We took on several of our own engineering students into a computing PhD programme. As previously mentioned, we noticed a significant skills and knowledge gap in the standard of Honours graduates between those students who have completed a three-year degree in England/Wales and those that have completed a four-year degree in Scotland and various Asia-Pacific universities. This was something of an eye opener and reinforced our realisation of the missing 'Unix culture' in our undergraduates.

The culture gap had a marked effect when we first established the computing laboratory and tutorial system. We found that not only did the undergraduates not get exposed to the 'usual' material, but there were no postgraduates available from whom they could learn.

In order to combat this gap we wrote a number of short courses that would rapidly educate our postgraduates in essential skills. We used a tutorial-based informal lecture system that focussed on small class sizes. As our students had a good (but basic) knowledge, the advanced concepts were taught mainly through the use of examples and having the students try to replicate and extend the taught material.

We now have no less than 12 excellent postgraduate students who are helping to pass on the computer science culture to new generations.

9 Filling in the Gaps

Understandably, employers from both the local area and across Britain want graduates with the highest degree of skills and knowledge possible. In order to address this skills gap, we have started a technology transfer centre within the university. This technology transfer centre, run by ourselves, runs essentially to up-skill our local graduates and graduates of other universities, equipping them with the programming skills necessary for them to be immediately effective in the workplace. Our industrial panel - a group of local business people interested in our department, motivated the development of this centre. They saw the extra need for training in specific packages and languages, as opposed to the general education in computing concepts as emphasised in our degree programmes. We run short professional courses that are not only attended by graduates in the area, but have also attracted employers and employees from across Britain to provide hands-on training for their research and development staff. The table below lists some of the short professional courses that we offer:

Core Java Programming	3 days
Advanced Java Programming	2 days
Introduction to JavaScript	3 days
Graphical JavaScript	2 days
C Programming	5 days
C++ Programming	5 days
Introduction to XML	2 days
Advanced XML: XPath and Stylesheets	1 day
Introduction to Extreme Programming	2 days
Perl Programming	2 days

Introduction to Unix	1 day
Introduction to PHP	1 day
Introduction to Statistics using SPSS	2 days
Multivariate Data Analysis with SPSS	2 days
Introduction to Databases and SQL	2 days
Web Server Configuration and Administration	3 days

Table 1: Short professional courses offered by our technology transfer centre.

These courses focus on specific skills and complement the more general skills taught in the formal degree programmes. We think it likely that many departments will need to keep a weather eye on focussed industrial needs and that providing courses like this is a palatable way of doing so. Needless to add, industrial consulting rates can be charged for such courses.

10 Resources

We have been able to build up the department to twelve full-time equivalent academics. This has been possible thanks to the support of our local government agency providing a boot-strapping cash investment in 2000. Undergraduate and postgraduate student numbers have almost doubled since then, enabling us to maintain a satisfactory level of staffing.

We have also been able to set up undergraduate and postgraduate laboratories. Our general philosophy has been one of diversity balanced against the support capacity maintainable with three full-time systems administrators. We have chosen to set up our main teaching laboratories using dual-bootable PC systems (Windows and Unix). We have supplemented this with a number of similarly-equipped break-out laboratories. Thanks to assistance from Sun Microsystems we have been able to equip one laboratory with thin-client workstations (SunRays). We are now contemplating a new laboratory equipped with Macintosh systems. On the server side, we have established a number of multi-processor and large disk-array hosting systems. Undergraduates also have access to: cluster computing systems; virtual reality and stereo graphics projection systems; a virtual private network; and wireless and mobile computing systems through the auspices of the research groups. At the time of writing, our workstation to full-time student ratio is approximately 0.6. We believe this is somewhat in excess of that provided by many computing departments in the United Kingdom.

11 Discussion

A number of high level issues have arisen from our opportunity to start from scratch. We present some of these here in the hope that they will be of interest to those involved in new and existing computer science programmes.

Possibly the most significant issue facing the design and implementation of a relevant and modern computing programme is the perceived "dumbing-down" of mathematics in secondary education. We have had no choice but to address this directly through custom-designed mathematics modules for our first-, second- and third-year students. The biggest burden is obviously on the first-year maths lecturers, who not only have to cover an increased body of material, but also have to face the traditional difficulties of bridging mathematical and theoretical computer science material with practical programming and algorithmics training. This is particularly exacerbated by the pressures on curriculum space; we have been forced to remove some the material we would like to cover in first-year to make space for this. We are not helped in this by the insistence of Quality Assurance Agencies and Accreditation bodies of explicitly covering issues like professional ethics, transferable skills and entrepreneurial and business process re-engineering issues. While we support the inculcation of such broad subjects into the undergraduate curriculum, it has been frustrating trying to do this under the constraints of a modular system.

Unlike the Australasian education systems, the UK has adopted the modular system at all levels of tertiary study. Although the modular system does enable the design of cross-disciplinary programmes, it is not without its difficulties. We feel these are particularly prominent in first-year study where the modular system is a positive disruption to producing a coherent introduction to computer science. We believe there is some merit in having a single first-year computer science course and only adopting the modular system in second, third and higher years for specialised teaching components.

Like many computer science departments we have had to face the challenge of which programming language to cover in depth and breadth over the course of the undergraduate degree. We have chosen to introduce Java at first year level as a compromise between an academically sound pedagogical teaching choice and a pragmatic means for graduates to gain immediate employment. We also expose students to C, C++ (which we believe covers imperative and object-oriented programming), and in addition specialist modules cover Scheme, Lisp and Haskell.

Although our School of Informatics was formed from Engineering, Mathematics and Computing for pragmatic resourcing reasons rather than academic ones, we do believe this mixture is a good one. The mix of programmes we are able to offer is useful for undergraduates to customise and delay their choice of specialisation until after they have entered University. It also provides, we believe, a fertile breeding ground for cross-disciplinary research. The number of engineering students who are now currently undertaking computing research evidences this. Of the approximately forty postgraduates in the School, nearly half of these are working on computing projects.

In spite of having limited teaching resource we do strongly believe that offering elective modules at second and third year adds to the distinctiveness and attractiveness of our computing programme. We have been guided by the ACM Model Curriculum in providing

a minimum set of options that we believe adequately cover the requirement of a modern international programme. We have chosen specialisms related to our preferred research activities including Distributed and High Performance Computing. We believe Artificial Intelligence is emerging from its "winter" and we are actively researching overlap areas such as autonomic computing and smart distributed systems. These are proving attractive areas for undergraduate projects and prospective postgraduate students. Although we have been surprisingly successful in attracting foreign postgraduate research students, we make the observation that for any successful computer science department some retention of local students is critical.

We have had considerable success in establishing links with local businesses. We have found an almost bottomless demand for graduate programmers from our programme. We have also established over £500,000 of industry collaborative funding based on teaching and research activities.

12 Conclusions

We conclude that, it *is* possible to start a Computer Science department from scratch. Computer science, as a discipline, is broadening and it is not possible to cover every topic. It is, however, possible to cover the core material, concepts and skills that constitute computer science. Having identified this, it can be successfully built upon to establish additional popular and relevant computer-related degrees.

It has been very challenging setting up a new computing culture but we believe once it is set up, its very momentum keeps it going. We have, in fact, seen this momentum spill over into other departments within the university.

We think it has been absolutely vital to have mathematical input to our degree programme. We make the important observation that all the computing technologies, from theoretical, to computer science, to the engineering hardware aspects, combine rather well to make an attractive portfolio of computing-related degrees. We cannot over-emphasise the importance of discrete mathematics to computer science and perhaps our biggest challenge has been building up recognition for this side of mathematics to complement material developed for continuous mathematics, which is used more in electronic engineering.

We have also found that it is not possible to consider undergraduate teaching in isolation from the remainder of the department. It is important to consider the complete departmental culture and how the undergraduate teaching links with the postgraduate training and research culture.

We have seen computing student numbers grow to 66 new students per annum in three years, with expectation of continued steady growth.

13 References

ASSOCIATION FOR COMPUTING MACHINERY. (2001): *Computing Curricula 2001.* Available from www.computer.org/education/cc2001

BRITISH COMPUTER SOCIETY. (2001): *Guidelines on Course Exemption and Accreditation.* The British Computer Society, January 2001.

MASKELL, D. & Robinson, I. (2001): *The New Idea of a University.* Haven Books.

MOSLEY, I. (ed) (2000): *Dumbing Down: Culture, Politics and the Mass Media.* Imprint Academic.

PINER, M-L. G. (ed). (2001): Defining Computing Curricula for the Modern Age. *ACM Computer Society Connection.* June 2001.

QUALITY ASSURANCE AGENCY FOR HIGHER EDUCATION. (2000): *Computing Subject Benchmark Statements.* Quality Assurance Agency for Higher Education.

14 Appendix

The following tables show our undergraduate teaching curriculum. The first year is designed to cover the basic topics of computer programming and discrete mathematics. The second and third years introduce more specialist core material and specialist options.

First Year = 10 Compulsory + 2 (any) options	
ICP1020	Introduction to Artificial Intelligence
ICP1021	Introduction to Databases
ICP1022	Java Programming 1
ICP1023	Java Programming 2
ICP1024	Algorithm Design
ICP1028	Information Systems
ICP1029	Computer Systems 1
IDM1015	Discrete Maths 1
IDM1016	Discrete Maths 2
IPS1080	Probability and Statistics
ASB1202	*Introduction to Banking*
IAL1032	*Algebra*
ICM1011	*Introduction to Communications*
IME1006	*Digital Circuits and Design*
IMM1001	*Mathematical Methods 1*
IMM1002	*Mathematical Methods 2*
PCP1002	*Perception and Cognition 1*
PCP1003	*Brain & Behaviour 1*

Table 2: First-year subjects. Optional subjects are represented in italics.

Second Year = 10 Compulsory + 2 (Informatics) Options	
ICP2011	Data Communications and Networks
ICP2022	Systems Software
ICP2024	Advanced Software Design Methods
ICP2025	Artificial Intelligence 2
ICP2027	Data Structures and Algorithms
ICP2030	Concurrency and Operating Systems
ICP2301	Project Planning and Software Hut 1
ICP2302	Software Hut 2 & Maintenance
IDM2015	Automata Theory
IDM2016	Complexity Theory
IAL2031	*Groups and Rings*
IAL2035	*Linear Algebra*
ICM2017	*Image Processing and Computer Vision*
ICP2026	*Computer Systems 2*
ICP2028	*Dataflow & Functional Programming*
ICP2029	*Theory of Languages and Machines*
IED2066	*Quality Value and TQM*
IED2069	*Entrepreneurship*
IES2005	*Digital Circuits and Systems*
IOR2085	*Operational Research 1*
IPS2005	*Data Analysis and Presentation*
PCP2001	*Perception and Cognition 2*

Table 3: Second-year subjects. Optional subjects are represented in italics.

Third Year = 6 Compulsory + Project (worth 3) + 3 (Informatics) Options	
ICP3022	Parallel Algorithms and Architectures
ICP3027	Database Management Systems
ICP3029	Data Networks and Distributed Systems
ICP3030	Graphics and Human Computer Interaction
ICP3099	Individual Project
IDM3004	Graphical Algorithms
IDM3015	Logic
IAL3007	*Abstract Algebra*
ICP3021	*Real-Time Systems*
ICP3028	*Neural Networks*
ICP3031	*Quantum Computation*
ICP3032	*Agent Technologies*
ICP3033	*Artificial Intelligence 3*
ICP3034	*Compiler Construction*
IED3064	*Business Process Re-Engineering*
IGT3001	*Computational Geometry*
IMM3050	*Wavelets*
IMM3061	*Numerical Analysis*
IOR3085	*Operational Research 2*
IOR3082	*Markov Chains*
IPS3083	*Statistical Pattern Recognition*
ICP4120	*Formal Methods*
ICP4121	*Modelling & Analysis of Distributed Systems*
ICP4122	*Computational Grid Systems*
ICP4123	*E-Commerce*
ICP4124	*Advanced Software Engineering*

Table 4: Third-year subjects. Optional subjects are represented in italics.