Managing Variations in Prior Learning
Related to Computing Skills

Word Count: 5160 (excluding references and appendices)

Topic Keywords: Prior Learning, Informatics, Ethics, Engineering Computing, Engineering Practice

Authors: A/Prof. David Lowe (Primary contact)
          Dr. Craig Scott

Affiliation: Faculty of Engineering
            University of Technology, Sydney

Address: P.O. Box 123 Broadway

City: Sydney

State/Province: NSW

Postal/Zip Code: 2007

Country: Australia

Telephone: +61 2 9514 2526

Fax: +61 2 9514 2435

E-mail: {david.lowe, craig.scott} @uts.edu.au
ABSTRACT

During 1997-8 the Faculty of Engineering at UTS undertook a major overhaul of its undergraduate programs. As part of this redevelopment we moved to a common core across all Majors (i.e. Mechanical, Civil, Environmental, Electrical, Telecommunications, Computer Systems, and Software). One of the core subjects - Informatics - focuses on an appreciation of issues surrounding various software and information tools. Two key factors influenced the design of this subject. First, the subject is offered to students across the entire engineering spectrum. As such, their motivation, interest and expectations of a computing subject vary significantly. The second factor is that computing technology is an area that currently shows very diverse prior exposure and skills. Our incoming students vary from having very sophisticated (though often narrow) skill sets, to barely having used a computer. The students are moving from a context (secondary school) in which they naturally develop extremely diverse prior learning and understanding to a context (university) in which we traditionally have had relatively uniform expectations regarding their learning outcomes. In this paper we consider the various approaches that we have adopted to cope with this diversity as flexibly as possible. We look at the structure of Informatics, the responses of the students, and how successful this has been in addressing the diversity of prior learning (related to computing technologies) that occurred in the transition from school to University.

INTRODUCTION

In 1997, the Faculty of Engineering at the University of Technology, Sydney (UTS) undertook a major redesign of its undergraduate Bachelor of Engineering programs. As part of this redesign a new core subject – Informatics – was introduced. This subject, which is taken by all Engineering students irrespective of the particular Major (Mechanical, Computer Systems, etc.) in which they will be specialising, develops skills in, and an understanding of, a range of computing and information tools. This includes technical aspects, such as introductory programming, the internet, spreadsheets, and databases, as well as the students ethical and professional responsibilities in using these tools appropriately. It is agreed within the Faculty that skills in these areas are critically important to all students.

The nature of computing technologies, and their utilisation within society, is such that the students entering the degree program have a very diverse set of knowledge and skills. This varies from students who have very sophisticated (though often narrow) skill sets, to those students who have little or no familiarity with computers or related tools and technologies. There is a noticeable gap (that appears to be widening is some respects) between those with exposure to computers and those
without exposure. It is also worth noting that there is a partial correlation between students’ Majors and both their level of prior knowledge and their motivation and interest.

The consequence of the above observations is that Informatics needs to be very flexible in adapting to variations in prior learning. Without an appropriate consideration of the variations and mechanisms to allow students to suitably adapt their study, the subject would fail to provide an effective learning experience for many of the students. In this paper we look at how the subject has been designed to accommodate the variations in prior learning, including both the underlying educational philosophy that we have adopted and the resultant structure, content and approach that we have developed.

In the following section, we discuss the background related to the redevelopment of the UTS Bachelor of Engineering, Diploma of Engineering Practice degree and the role that the core subjects play within the structure of the course. We look at the contribution of Informatics to this structure, and the graduate attributes of the course overall. We then consider the variations in prior learning of the students and how this might influence various aspects of the subject. In the next section of the paper we look at how we have designed the subject to address the variations in prior learning. In particular we consider both the structure of the subject and the way in which the logistics of the subject are managed to provide flexibility whilst ensuring that appropriate learning outcomes are achieved. We then go on to look at the results of our approach and some experiences in the first few offerings of the subject. Finally we consider how we might continue to improve the subject, along with some conclusions regarding the approaches that we have adopted.

BACKGROUND TO SUBJECT DEVELOPMENT

Bachelor of Engineering degree program

The previous Schools of Civil, Electrical and Mechanical Engineering at the University of Technology, Sydney (UTS) had, since 1972, each offered programs leading to the award of a BE degree in one or more specialist engineering disciplines. These courses had various common features, such as a commitment to co-operative education, but differed substantially in structure, teaching methods and expected learning outcomes. This has happened despite the consensus across the profession that there exists a significant body of knowledge, skills and values which are held in common by all professional engineers, whatever their specialist discipline.

Supported by the arguments and recommendations of a plethora of reports on engineering education published during the mid-90’s, culminating in the Australian Review of Engineering Education [2], the Faculty came to the view that the separate BE programs should be jointly redesigned as a unified degree with the option of specialist majors. The new program, which was designed during 1997, and introduced as a combined BEDipEngPrac (Bachelor of Engineering, Diploma of Engineering Practice) degree progressively during 1998 and 1999, addressed the shortcomings that were raised in the above reports. These shortcomings included: inadequate attention to the professional formation of students; curriculum fragmentation and overcrowding; and insufficient opportunities either for students to cross traditional disciplinary boundaries or for courses to adapt to changing external demands.

One of the key design objectives for the new course was to explore how to accommodate diverse learning pathways that were adaptable to students’ individual circumstances encompassing both guided and independent study either on campus, at home, or in the workplace. Such flexibility has the potential to improve learning, increase convenience, and reduce completion time. The resultant degree program contained various elements. One of the key ones was a set of core subjects common to all majors in the undergraduate Bachelor of Engineering degree, with the students focusing their
degree through both subjects specific to their choice of a major (discipline) and electives which allow either further specialisation or a broadening of experience and knowledge (based on student preferences).

The core includes subjects covering aspects such as engineering communications, project management, engineering finance, uncertainties and risk, and – of course – computing. The computing aspects are predominantly covered in the subject Informatics. Being a core subject, Informatics is undertaken by all students irrespective of their field of practice - be it Mechanical, Civil, Environmental Systems, Electrical, Telecommunications, Software, or Computer Systems. This has several implications – such as large variations in the skills, prior learning, expectations and motivations of the students taking the subject. We shall consider these points in more detail shortly, but first let us consider the focus of Informatics.

**Informatics**

The aim of Informatics is to introduce 1st year students to the use of computing tools in the context not only of engineering problems (which is the way computing subjects are often taught in Engineering departments) but in the context of engineering practice. For example, software development subjects will often teach programming in a way that shows how it can be used to address specific problems; such as how to write software for engineering analysis. Such subjects will, however, rarely consider issues such as the impact of errors or inaccuracies in programs, the professional responsibilities of the students with respect to the developed software, or how students develop an ability to continually adopt changing tool sets and development environments. In developing Informatics, we explicitly attempted to address these considerations.

Another aspect that Informatics needed to consider was the rapid pace of technological change. Most computing courses fail to recognise both that the range of tools that students will encounter in engineering practice goes way beyond what can be even partially addressed within a formal degree program, and that many tools are changing so rapidly that there will be little similarity between those available at the start of a student’s degree and those at the end of the degree. This means that the students will regularly encounter tools with which they have no experience or familiarity. Informatics therefore needs to ensure that students develop an ability to evaluate and learn new tools.

Further, students often lack the ability (or confidence) to continually develop their own skills. Yet as part of their professional engineering practice they will be constantly required to develop not only their knowledge, but also their skills in utilising the tools that support them in their professional practice. This means that the students need to learn not only specific tools, but need also to “learn how to learn” so that they may continually refresh their existing tool skills and acquire new ones without the support that a teaching and learning environment provides. This point is particularly critical in the context of a flexible approach to learning in the subject. Again, we shall return to this point shortly.

Given the above considerations, Informatics was developed with three core aims. The first is to develop specific skills in using information and computational tools (such as spreadsheets, the internet, and programming languages). The second is to develop an ability to continue to develop skills in a broad range of tools. The third, and most unusual, is to develop an understanding of the professional issues that surround the use of computational tools - the types of tools and resources that allow engineers to manage the information, models, complex calculations, and processes associated with carrying out engineering activities.
Student profile

As was mentioned above, *Informatics* is taken by all Bachelor of Engineering undergraduate students irrespective of their particular major - mechanical, civil, environmental, electrical, computer systems, software, or telecommunications. This has several immediate implications. The first is that *Informatics* is the only computing subject to be undertaken by many of the students, whereas for others it is only an initial introduction prior to an entire thread of software subjects. This means that there are relatively complex requirements on the material that needs to be covered and the skills that should be developed within *Informatics*. It also means that the motivation of students varies considerably. Some of those students for whom this is the only computing subject see it as either unnecessary or are rather uninterested – especially in the ethics sections of the subject. For others, especially those for whom it is the first in a sequence of computing subjects (for example, telecommunications, computer systems, and software engineering students) they are more highly motivated and interested in achieving the maximum benefit from the subject. In each case, the students hold very different expectations of what they wish to achieve in the subject and their understanding of their own learning objectives.

Even more significant than variations in motivation, and the key focus of this paper, is variations in prior learning of the students. We are taking students from a schooling background where they have naturally developed extremely diverse prior learning and understanding, and placing them into a context where we traditionally have had relatively uniform expectations. Although we have long experienced a very broad diversity in the cultural and social backgrounds of the students entering our degree, the educational diversity is restricted by the entrance requirements (e.g. 3 unit HSC mathematics). We would claim that one of the most variable elements of prior learning (of those that relate directly to course material or expected learning early in the undergraduate programs) is the students familiarity with computing concepts. This is partly due to the variations in access to computing technology, partly due to changing social expectations regarding familiarity with computers, and partly due to the lack of well-defined entrance criteria in this area (at least in the same way that we have well-defined expectations regarding English language, science and mathematical abilities).

As an example of the variations that exist, consider the data shown in Table 1. Several interesting observations can be made. For example, with specific skills (such as programming, spreadsheets and databases) there is a very strong bimodal distribution. Students in general appear to either be quite familiar with these, or very unfamiliar. This is in contrast to a technology such as the Internet, where the spread is much more even. Table 2 also shows that there is a relatively strong correlation between level of prior knowledge (as measured by the self-assessed familiarity with various technologies and tools) and intended engineering discipline.

| Table 1: Variations in students' familiarity with a range of tools and technologies (source: student surveys carried out at the commencement of the Autumn 1998 semester) |
| Table 2: Variations in students' familiarity across different engineering programs (source: student surveys carried out at the commencement of the Autumn 1998 semester) |

In the initial discussions regarding the subject prior to its development, it was made very explicit that *Informatics* needed to actively accommodate this variability in prior learning. It is also important to note that, as discussed above, a key goal of the subject was to develop the students' ability to not only use specific skills but to also continue to develop new skills in using computing tools. This goal meshes very well with the need to accommodate variability in prior learning. If we are able to develop this meta-
skill, then the variations become much less relevant. If we are able to achieve the subject goals then where students do not have specific required skills then they should be in a position to be able to develop such skills themselves. This then means that the focus of the subject can shift to a combination of developing these required meta-skills and providing the students with the flexibility to extend their own learning relative to their prior knowledge. One factor worth mentioning that impacts on this is that there are also pragmatic constraints – for example, we need to ensure that students have the pre-requisite skills necessary for success in later stage subjects.

SUBJECT DESIGN

Development of Informatics

Let us now consider how the design of the subject has addressed these specific requirements. A consequence of the fact that Informatics is undertaken by all engineering undergraduates (and hence has both great diversity and large student numbers) is that it has rather complex educational requirements. In order to ensure that these were appropriately considered the development team for Informatics adopted a very structured approach to the subject development.

We began the development by identifying both subject constraints and resourcing issues, and the specific educational requirements. The subject requirements were constructed by initially selecting a subset of the graduate attributes defined for the overall BE degree programs. From these graduate attributes we identified the key educational objectives and hence formulated the subject requirements. For example, one of these requirements was:

Requirement 6) Informatics must provide the opportunity for all students to develop their knowledge and skills irrespective of their level upon commencing the subject.

We then designed the subject in a way that specifically addressed the identified requirements. Full details of the design of the subject and the way in which addressed skills development can be found in [3,4]. Of particular interest in this paper is the way in which we addressed the issue of variations in the prior learning of students.

Flexible structure: Adapting to student expertise

A common approach for coping with variation in prior learning is to develop a bridging or remedial course (or several in different areas) that brings those students with minimal experience up to a level consistent with the subject entry requirements. This approach is appropriate where there is a small proportion of the students who lack relatively well-defined skills or understanding. In our circumstance, however, the diversity of understanding is such that the range and structure of bridging courses that would be required would be prohibitively complex.

Rather than having a remedial or bridging course we have taken a very different approach. Because the particular expertise of students varied considerably, with different students having very different prior learning, we developed the subject to have a very modular structure. (Actually, the modular structure was also a consequence of various other factors, such as the difficulty of appropriately resourcing a computing subject with in excess of 500 students per year)

Essentially, the subject was designed around a series of subject modules. These modules focus on different aspects of the subject, and are broken into two main streams: skills and issues. The issues modules cover the development of understanding of classes of informatics tools, uses and limitations
of these tools, the relationships between these tools, meta-skills required to independently learn new tools, and ethical and professional obligations in using these tools. The skills modules cover specific skills in different tools, including spreadsheets, the Internet, operating systems, programming, and a number of other tools. It is in the skills modules where we encountered the greatest diversity in prior learning. An extract of the subject design documentation, detailing much of this development, is given in Appendix 1.

Students who are undertaking Informatics are required to complete a set of compulsory modules (the majority of the subject), and one or more elective modules (to demonstrate their ability to independently learn new tools, and to extend their skill set). To accommodate the huge variability in the prior learning and experience of the students, and the variations in student expectations, the students are given the option of negotiating to replace compulsory modules in which they can demonstrate appropriate prior understanding with an alternative selected from the elective modules. In this way we are encouraging students to utilise the opportunity to extend themselves and maximise their learning as much as possible. This also provides them with the opportunity to focus their learning in areas most appropriate to their needs. The consequences of this are that student expectations and motivations are positively influenced concomitant with positive changes to study strategies and learning outcomes [7].

Note also that to provide a sense of cohesion to the subject (especially in light of its module-based nature), students are allocated to a specific tutor for the duration of the subject. This tutor takes the students for all small class sessions, including both issues and skills modules. The tutor becomes responsible for the students overall subject program (including aspects such as negotiating with them about replacing compulsory modules with electives) and will normally be the students first point of call in the case of questions or problems related to subject content and logistics.

One concern that we had in designing this structure for the subject was that students would choose the “easy option”. Rather than electing to undertake a module with which they were unfamiliar (and required substantial learning effort) they may choose to simply keep quiet and accept a default module with which they were already highly familiar – and in which they were more likely to obtain a higher assessment grade. We addressed this concern in several ways. The first was to get the tutors to take an active role in discussing with students their particular study pattern in the subject. The second was to modify the subject assessment so that it minimised the benefit to students of completing modules with which they were already familiar – the focus of assessment was shifted away from the marks achieved and more squarely onto learning outcomes.

The assessment was structured such that the subject incorporated both Mastery and Advanced assessment levels. All students are required to complete the Mastery assessment component, which solely determines each student's pass/fail grade but not their final mark. If the student passes the Mastery component then they are guaranteed a pass result in the subject even if they do not attempt the advanced assessment components. If they fail any component of the Mastery assessment then they fail the subject. This approach removes the common problem of “mark-accumulation” and focuses the student on developing an appropriate level of understanding. To satisfy the Mastery requirements, the students must satisfactorily complete all the subject modules (including any negotiated elective modules), and pass the final oral and coding exams. Satisfactory completion of each module is based on evaluating, on a Pass/Fail basis, the student achieving an acceptable level of understanding. This is determined through the module assessment tasks and through the examination of a Journal that the students maintain throughout the subject. The Journal is an important element of the assessment and the students are encouraged to record their reflections as they engage with the subject matter. This is important in determining the extent to which they have extended themselves by
undertaking modules that cover new material. Those students who wish to earn a grade beyond a Pass can attempt the advanced assessment component. The Advanced assessment components involve a major programming assignment and a learning contract. The learning contract requires the student to negotiate a task that demonstrates understanding of two or more of the non-programming skills modules and at least one major aspect of the issues modules.

The Mastery/Advanced approach to assessment has been used extensively in similar subjects (notably software development subjects) over the last few years and has proved to be particularly effective [5]. This structure encourages students to focus on their learning, as distinct from focusing on mark accumulation, yet provides the scope for more ambitious students to obtain a grade which reflects the extra learning and understanding they have achieved. This approach also allows tutors to concentrate on constructive feedback rather than focussing on allocating marks in situations where they matter little. The negotiated Learning Contracts that form the Advanced assessment allow students the freedom to explore topics and problems which are of particular interest to them. The diversity of Learning Contracts negotiated and submitted is testament to the motivational value of this approach. A compulsory component of any Learning Contract is a personal reflection examining the degree to which the student met their learning objectives, which approaches to the learning goal(s) were effective and why, and also those approaches that were less effective.

RESULTS AND EXPERIENCES

At the time of writing this paper Informatics has just completed its third offering. During this time we have gained substantial feedback on the effectiveness of the approaches which we have adopted. This feedback has been obtained from informal discussions with students, formal subject evaluations, small group discussions, and student reflections included in the students’ journals. Based on this feedback, we can make various observations about the success (or otherwise) of our approach.

The first and most obvious observation is that, in general, the students have shown a high degree of enthusiasm for the subject. In particular the technical nature of the much of the subject gains the interest of the students – especially when compared to the more theoretical subjects that the students typically encounter in their first year of University. Comments such as the following are relatively typical.

“I came to University expecting to be able to do things – and this is the only subject where I got to do this.”

“The programming and internet bits were good but I didn’t like the ethics stuff – I got enough theory in other subjects!!”

Student feedback has also indicated that the overall structure of the subject is appreciated since it offers considerable choice in both what is learnt and how it is assessed. For example, two students commented that:

“the electives that are offered are helpful – they allowed me to learn things that are most relevant”

“[the various modules] really challenged my existing knowledge”.

This choice has had a quite positive influence on the student motivation.

Despite this positive impression of the content and structure of Informatics there have been several negative aspects. The first is that the students found the logistics of the subject rather complicated.
This especially related to the selection of elective modules and keeping track of what this selection meant in terms of attendance, assessment, etc. For the subject flexibility to have maximum positive effect the students need to be able to understand the structure of the subject, how the various components interrelate and the expectations on them. This also led to students having difficulties in managing their workload. Whilst subjects involving a programming component typically have a heavy workload, the modularised structure of the subject meant that each individual module was subject to "content bloat". It also exacerbates difficulties which students typically have with time-management – a particular problem during their first year at University when they have much more freedom (both socially and intellectually) than that to which they are accustomed.

We are currently considering ways in which these difficulties can be addressed. First, we will be endeavouring to provide better guidelines as to how much work they should have completed at any stage in the semester and to assist them in managing their time. We are also considering ways in which we can improve the synergy between modules to control the workload expectations without making the logistics even more complicated. The simplest approach, and the one that can be implemented immediately, is to introduce one or more key problems and thread these through many or all of the modules. This will allow more direct comparison and evaluation of the various tools and issues arising from their use. This will also address some students’ interpretations that the modules are isolated and unrelated components, as well as reducing the work load as students are not learning new problem domains.

Even more significant however was that our attempts to recognise prior learning were not as successful as hoped. Relatively few students elected to opt out of a mastery component with which they were already familiar and undertake an extra elective module. For example, consider the data shown in Table 3 (from the Autumn 1998 semester).

Table 3: Numbers of students who negotiated subject module changes based on prior learning (source: Autumn 1998 student results spreadsheet and student records).

There are likely to be many reasons for this unwillingness to negotiate a change in modules based on prior learning, but a strong disincentive for the students is the scheduling of the electives. These were scheduled towards the end of semester when the students are busy with the advanced components of the subject as well as with all their other subjects. This can be overcome relatively easily by revising the schedule and running elective modules throughout the semester. This latter change may be supported through the development of self-learning packages.

We have used self-paced learning modules in both Informatics and other computing subjects to teach aspects of programming. Typically these learning modules are set out as a sequence of precise instructions for the students to work through – including readings, discussions, exercises, and material to be submitted to the students tutor for feedback or assessment. The students’ response to these learning modules has been excellent, particularly in terms of the flexibility that they have provided to students in adjusting their pace and style of learning to their particular needs.

One avenue for using learning modules more effectively that we are currently investigating is to change the assumed baseline understanding in the subject. At present the default program for students is to undertake all basic modules, but allowing students to negotiate changes based on prior knowledge. Unfortunately, many students elect to take the easy option (through apathy, being unaware of options, or unsure of the processes for changing their program) and not negotiate any changes when they are
definitely suitable. A possible change would be to remove the baseline set of modules and require all students to select a set of subject modules from a pool of modules. Their selection would be based on their prior knowledge. To ensure that we achieved a minimal level of understanding for all students we could construct a well-defined set of subject outcomes that all students would be expected to achieve. The students' selection of modules could be supported by providing all modules with a self-test which could be taken at the beginning of the semester by students to allow them to determine whether that module would be beneficial.

The self-test concept will also help address another problem with respect to student selection and/or negotiation of modules. We have found that in a number of cases the students were unable to effectively evaluate their own level of understanding with respect to a particular topic, technology, or tool. For example, many of the student had some brief experience with writing short simple software programs, and assumed that this meant they were familiar with (or at least able to write) much more complex software - an observation which has no support in current practice. By providing self-tests we would assist students in more critically examining their own knowledge and skills, and thereby being able to make a more informed decision about their selection of modules.

Finally, it is important to note that the diversity of coverage in the subject (ranging from various tools, to considerations of the students' professional obligations in using these tools) was seen as a strong positive factor. A typical comment was “The best aspects were that we got a good introduction to many applications which we will use throughout our education/career”. Subject surveys and anecdotal evidence indicate that the students are developing, very early in the degree, a sense of the context that defines how they can (and should) approach the practice of engineering in a professional context.

CONCLUSIONS

In this paper we have considered an approach to managing the diversity of computing experience exhibited by first year engineering undergraduates. Our approach is based on a combination of a modularised structure, student-negotiated selection of modules, and mastery-advanced assessment. The approach has proven to be very effective when students are motivated to undertake the negotiation. It however has problems when students do not have sufficient motivation or understanding to negotiate changes. This has unfortunately occurred with a rather large proportion of students.

Initial investigations have shown this to be at least partly because the students are not familiar with the increase in academic freedom which comes with the change from School to University. We discussed one possible solution involving making the negotiation of subject modules to be completed an integral part of the subject rather than an option. This has, however, yet to be trailed.

Overall, the subject has proved to be very rewarding for both staff and students. Creative ways of structuring the subject, flexibility in the student learning approaches, and providing an engineering practice context for the development of understanding computational tools, have all resulted in a valuable and effective learning experience. Overall, the subject has been logistically fragile, but educationally successful.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the invaluable contributions of other staff involved in the development and implementation of Informatics: Martin Evans, Ravin Bagia, James Boswell, Ron Meegoda, Austin Mack, Richard Lai, Ben Rodanski and Ali Saleh.
REFERENCES


APPENDIX 1: SUBJECT DESIGN DOCUMENT EXTRACT

The following information is an extract from the Informatics subject development documentation describing (in part) the initially proposed subject structure. This highlights the way in which Informatics provides a flexible structure which allows students to select learning modules that are best suited to their particular prior knowledge and skills.

Subject Structure

... The subject structure involves two main threads of focus: issues and skills. The issues thread (equivalent workload approximately 48 hours) covers the development of suitable understanding of classes of informatics tools, uses and limitations of these tools, and the relationships between these tools. The skills thread (equivalent workload approximately 112 hours) covers skills development.

The subject is structured so that it is comprised entirely of teaching modules. Some of the modules will be compulsory (especially the issues modules and the several of the skills modules - including programming skills) and some of the modules will be electives. The issues modules will be scheduled so that they run essentially throughout the semester.

The formal sessions for the various teaching modules includes large classes (lectures involving presentation and explanation of material to the entire subject cohort), small classes (either lectures involving presentations or workshops involving interactive discussions with smaller tutorial-sized groups of students), individual study (tutor-supported study involving one-on-one interaction between a tutor and a student such as might occur in the LDC) and independent study (study undertaken solely by the student, including both LDC modules and personal study). Note that each of these modes of learning/teaching has different resource implications.

During the first week of the subject, students will be allocated to a specific tutor (who teaches them during any small class sessions for the issues modules - see the section below on the issues thread) for the entire duration of the semester. This tutor becomes responsible for the students overall subject program (including aspects such as allowing them exemptions on compulsory modules and maintaining their marks for the subject) and will normally be the students first point of call in the case of questions or problems related to subject logistics.

Learning Modules

Each module is run relatively independently and will be structured in whichever way is considered most relevant for that module. The students will be required to complete a minimum of 19 units of modules. A unit corresponds to approximately 8 hours of total time commitment - which will include the option of two 1.5-hour class sessions, and related self-study time (though the specific distribution of time can vary from module to module, for example, module B may be a two unit module with 4 class sessions and 10 hours of self-study, and module B may also be a two unit module but be entirely based around LDC modules with no formal class sessions).
The students are required to complete a minimum of 19 units, (though most will end up completing 20 units because of the structure of the units), made up of both certain compulsory modules and additional elective modules. Where students have prior understanding or skills, compulsory modules may be replaced by the elective modules (obviously at the discretion of tutors or the subject coordinator). An example suite of skills modules is:

Compulsory modules (18 units):

All of the following issues modules (6 units):
- I1 Informatics Tools (3 unit)
- I2 Informatics Issues (3 units)

All of the following skills modules (7 units):
- CO Operating systems (1 unit)
- CW Written communication and word processing (1 unit)
- CS Spreadsheets (2 unit)
- CI Internet tools and the WWW (1 unit)
- PI Intro to programming (2 units)

And one of the following (note: the specific languages offered may vary from semester to semester and will be dependent on a variety of factors. Those listed are simply a typical set):
- PC C programming (5 units)
- PB Basic programming (5 units)
- PF Fortran programming (5 units)
- PJ Java programming (5 units)
- Px User-selected language (n units) note: this incorporates a student-selected language, negotiated with the subject staff and dependant upon resourcing implications.

Elective modules: both skills and issues modules may be offered (each module is 2 units).
- ET Teamwork tools
- EO Oral communication tools
- ED Databases
- EP Project management tools
- EC Computer Aided Design
- Ex Self-Learning module (student-nominated field)

+ others as developed or required

Each student will be able to select their particular pattern of modules. Each module will be commenced by a student by collecting the module documentation from the Learning and Design Centre. This documentation will describe the completion requirements for the particular module. This will include a combination of class sessions (up to two 1.5-hour sessions per skill unit), learning modules, self-study, and group projects.

Where the module contains formal class sessions (either lectures, tutorials or workshops) they must register for one of these sessions. The sessions will generally be scheduled to allow students to complete any combination of modules. Each small class session will be run by one tutor and have a maximum of 30 students. As an example, the spreadsheets module (2 units - 16 hours) may comprise 4 teaching sessions (6 hours), 4 self-taught modules (8 hours) and some self study (2 hours).

One introductory sessions will be held at the beginning of the semester. In this session the various modules will be described and the structure and approach outlined. At the end of the session, the students will be asked to provide an indication of the modules which they are likely to wish to undertake (including if they believe they will obtain permission to replace a compulsory module with an elective module - though this must later be confirmed by a tutor).

... Assessment

With the described subject architecture it is impractical for every module undertaken by a student to have individual assessment tasks, as this would result in an excessive number of assessment tasks within the subject. Nevertheless it is important that the extent of the students understanding of both skills and issues be effectively evaluated.

One possible proposal for the assessment for the subject would be to incorporate Mastery and Advanced assessment levels.

All students would be required to complete the Mastery assessment component, which solely determines the subject pass/fail grade (but NOT mark). No marks are allocated to the Mastery component, but if students pass then they are guaranteed a minimum mark of P (i.e. they are guaranteed a pass in the subject, even if they do not attempt the advanced assessment component). If they fail the mastery component then they are given a mark of Z. To satisfy the Mastery requirements, the students must:
Complete satisfactorily 19 skills units (including all compulsory units unless given exemptions). The definition of satisfactory is module defined.

Pass the final oral/coding examination (which examines their understanding of both issues and skills - i.e. questions may be asked relating to any of the skills modules they completed).

Those students who wish to gain a mark beyond a basic pass can attempt the advanced assessment component. The advanced assessment component (of which each and every element is completely optional) determines the students mark once they have passed the mastery criteria. Note that irrespective of performance in the advanced assessment component, students CANNOT pass the subject without passing the mastery assessment component. The advanced assessment components are:

50 marks: The assessment task from the programming module which they undertook.

50 marks: A learning contract (or similar assessment task) which demonstrates understanding of two or more of the non-programming skills modules and at least one major aspect of the issues modules.

Note that those students who obtain an exemption from the programming module (due to prior knowledge) can still undertake the assessment task from the module for which they have been given an exemption.

Additional Comments

A few additional comments are useful. This proposed architecture not only addresses resourcing issues elegantly, but provides a convenient and effective mechanism for coping with the disparate entry-level skills and understanding of the students. If a student already has a solid grasp of the material for a specific (compulsory) skills module (evaluated through an appropriate mechanism prior to commencing the module) then they can remove that module from their program and add extra elective modules, covering skills which they do not yet have.
APPENDIX 2: EXAMPLE FRAGMENT FROM SELF-PACED LEARNING MODULE

The following is an extract of one of the self-paced learning modules. This demonstrates the key elements and how they guide students through a particular body of knowledge.

Learning module 2

Objective
To develop your understanding of, and ability to code C programs incorporating complex data structures (pointers, arrays, structures, etc.), functions (especially passing of arguments, and passing by reference), the C libraries and the C preprocessor (including an understanding of header files).

Step 1: Arrays

Discussion:
As was covered in the first Learning module, programs utilise variables to store and manipulate data. Often we will have a collection of data which has the same type. For example we may want to read in and then analyse a group of 100 numbers. Rather than having to define 100 different variables, instead we define an array. This is done as follows:

```c
main()
{
    float i[100];
    // defines an array variable called i
...
```

[Material removed]

Reading:
Deitel and Deitel: Section 6.1 to 6.4

Action:
Try writing a short program which asks the user for a number of integer marks. As the numbers are entered, store them in an array. When the user enters a negative number, stop entering data (hint, try using a do-while loop) and calculate the average of the numbers. The program output should look like:

```
Please enter a mark > 56
Please enter a mark > 71
Please enter a mark > -1
Average is 63.500000
```

[Material removed]

Submission:
The file `mod2-8.c` contains the start of a program which will play tic-tac-toe (or "noughts-and-crosses") with the user. You are required to complete the program. Please note that you should leave the structure of the program unchanged and only add the sections specified. Where you are required to add type definitions, these should be consistent with the parts of the program which already exist. Finally, you can assume that the user always plays first (and is "O") and that the computer (which is "X") does not need to be particularly good at playing (i.e. it can play by simply finding the first available empty square).

Once you have completed your program then you should compile and test the program. After you are satisfied with your solution then you should email your working (and suitably commented) source code to your tutor for evaluation.
Table 1: Variations in students' familiarity with a range of tools and technologies *(source: student surveys carried out at the commencement of the Autumn 1998 semester)*

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computers</td>
<td>4</td>
<td>3</td>
<td>21</td>
<td>29</td>
<td>18</td>
</tr>
<tr>
<td>Microsoft Windows based</td>
<td>6</td>
<td>5</td>
<td>21</td>
<td>28</td>
<td>15</td>
</tr>
<tr>
<td>Macintosh computers</td>
<td>59</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Any computer programming</td>
<td>31</td>
<td>6</td>
<td>6</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>language</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>33</td>
<td>8</td>
<td>7</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>Databases</td>
<td>42</td>
<td>2</td>
<td>6</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Internet / World Wide Web</td>
<td>10</td>
<td>14</td>
<td>31</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>Email</td>
<td>12</td>
<td>8</td>
<td>7</td>
<td>12</td>
<td>36</td>
</tr>
</tbody>
</table>
Table 2: Variations in students' familiarity across different engineering programs *(source: student surveys carried out at the commencement of the Autumn 1998 semester)*

<table>
<thead>
<tr>
<th>Engineering Field</th>
<th>Average Familiarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Civil Engineering</td>
<td>1.97</td>
</tr>
<tr>
<td>Computer Systems Engineering</td>
<td>3.88</td>
</tr>
<tr>
<td>Electrical Engineering</td>
<td>3.13</td>
</tr>
<tr>
<td>Mechanical Engineering</td>
<td>2.43</td>
</tr>
<tr>
<td>Telecommunications</td>
<td>3.65</td>
</tr>
</tbody>
</table>
Table 3: Numbers of students who negotiated subject module changes based on prior learning (source: Autumn 1998 student results spreadsheet and student records).

<table>
<thead>
<tr>
<th>Description</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of students</td>
<td>259</td>
</tr>
<tr>
<td>Number of students estimated to have prior learning consistent with at least one module</td>
<td>134</td>
</tr>
<tr>
<td>Number of students estimated to have prior learning consistent with at least two modules</td>
<td>43</td>
</tr>
<tr>
<td>Number of students who discussed a module change</td>
<td>54</td>
</tr>
<tr>
<td>Number of students who undertook a module change</td>
<td>32</td>
</tr>
</tbody>
</table>
BIOGRAPHIES

A/Prof. David Lowe
BE PhD MIEEE MACM MAAEE MAACE

David is an Associate Professor, head of Computer Systems Engineering, and program director for the Information Systems Engineering graduate program at the University of Technology, Sydney. He has active research interests in the areas of hypermedia, multimedia and the Web. In particular he focuses on information contextualisation, hypermedia development process modelling, and web project specification and scoping. He has published widely in the area, including a graduate-level text (Lowe and Hall, "Hypermedia and the Web: An Engineering Approach", Wiley, 1999). He has consulted widely in the areas of web systems and software engineering, as well as running industry short courses in these areas.

Dr. Craig Scott
BE, PhD

Craig is a Senior Lecturer in, and Program Director of the Computer Systems Engineering Program at the University of Technology, Sydney. He teaches predominantly in software development but he also coordinates the orientation camp for first year students. Craig’s research interests are focussed on positioning systems, especially positioning cellular mobile phones. In particular, he is interested in techniques for modelling and then integrating the many sources of information that can augment the radio-location process.