Portfolio

Michael Jones
Bournemouth University

June 2006

Contents

1. Portfolio Objective

This portfolio was constructed in June 2006 as part of the Disciplinary Commons in the Initial Teaching of Programming. The contents of the document form a personal statement, and should not be seen as representing Bournemouth University in any way.

The objective of the document is to illuminate the experience of designing, implementing, and reflecting on the process of leading a Programming unit in an undergraduate Computing programme.

The initial sections describe the University, the unit, and the resources. Later sections cover the reflection on the main aspects of the unit.

2. Abstract

The first year Programming unit in the Computing Undergraduate Framework at Bournemouth University is delivered to a cohort numbering about 75. Delivery involves a single one hour lecture per week throughout the year, augmented by a weekly three hour workshop for each of group of 15 students. It is assessed via a combination of equally weighted coursework and examination.

All of the students have taken related subjects prior to joining the course, but the nature and level of this study pertaining to programming varies considerably. In addition, some students have pursued computing as a hobby. The result is a wide range of ability, and a number of students with no real comprehension of what programming involves.

This document provides descriptions of the content and context of the unit; includes examples of documentation; and relates the experience of delivering the unit during 2005-06, the first year of the revised programme.

Innovations introduced included more use of software to speed up the production of feedback for the students, and the use of oral feedback to provide more detail than traditional written feedback affords.

Weaknesses observed included a lessening of student participation in the latter stages of the year, and insufficient opportunities for feedback from students. Formative feedback to students was also less evident than would be ideal.

3. Bournemouth University

Established as Bournemouth College of Advanced Technology in 1973, the organisation developed rapidly in the late 1980's and early 1990's. This period saw the name change four times in the space of six years, becoming one of the 'new' universities in 1992.

This development saw a rapid expansion in full-time student numbers from 2,800 in 1988 to over 15,000 today (including partner colleges), of which some 10% are postgraduate. The curriculum has also undergone a transformation. Prior to 1987, the Dorset Institute (as it then was) focused on vocational, part-time National Diploma

courses and preparing students for external University of London examinations. In the period 1989 to 1994, an average of 20 new courses were introduced each year.

The University is mostly situated on the Talbot Campus, which is actually in Poole, hence 'Bournemouth University', not 'University of Bournemouth'. The other, smaller campuses are based in the centre of Bournemouth.

The large majority of the students are UK nationals, drawn primarily from the M3/M4 corridor.

A selection of images of the Talbot Campus are included in Appendix A.

3.1 The Current Climate

The university has operated a tight control over financial matters, so has a smaller deficit than many in the sector. A new Vice Chancellor took over in the summer of 2005, and has begun a programme of radical change. A voluntary severance scheme is in place, as is a programme of recruitment for 80 PhD students. The clear emphasis (and stated aim) is to shift the main goal to the achievement of quality research output.

4. The School of Design, Engineering & Computing

The School of Design, Engineering & Computing (DEC) is one of six Academic schools, and has approximately 1650 undergraduate and 100 postgraduate and research students. Within the School, Computing accounts for approximately 50% of all undergraduate and 40% of postgraduate students. There is also a substantial computing presence in other schools, notably the Media School, which includes the (RAE 5) National Centre for Computer Animation.

All the computing students and staff in DEC are based on the Talbot Campus.

The staff:student ratio in Computing within DEC is approximately 1:25.

4.1 Programmes in the School

The university has retained a programme-oriented, non-semester-based delivery of units. A few units are shared between programmes. The typical structure of a programme in DEC is 6 units per year for the first 2 years, with 4 units plus a double-weighted project in the final year.

All the computing courses in the School are sandwich programmes, with a third year spent in a related vocational role. We have had few problems placing students.

Employment of computing graduates is monitored, and typical figures are 85% of graduates employed in computing within 5 months of graduation. A few move on to research or postgraduate education.

5. University Facilities

The following refers to the facilities available at the main (Talbot) Campus.

5.1 Usage

The utilisation of the space resources in the University is the highest in the sector, by some 40%. The new Vice-Chancellor has indicated that there will be a building programme to "rectify this" (his words).

The campus laboratories are available on a 24/7 basis. The only exception is the Christmas shutdown of about a week.

5.2 Lecture Theatres

There are nine lecture theatres, ranging in capacity from 45 to 250. All have appropriate AV equipment, including a computer and a single data projector. Given the level of usage of space resources, there is a high level of contention for lecture theatres.

5.3 Programming Laboratories

The University generally has a high student to computer ratio, which is encouraging. However, the room utilisation is such that, according to students, finding a spare programming laboratory during the day (between scheduled sessions) is virtually impossible. There are general workstations available in the Open Access Centre and the Library, but these do not have the same image as those in the Computing laboratories.

As of Autumn 2005, the Talbot campus is fully wireless-enabled, which mitigates this, for those with (and willing to bring) laptops.

Programming laboratories typically seat 18 students, and are equipped with data projectors. One member of staff will supervise a laboratory session. All but one of the computing laboratories use Windows - there is a single Solaris lab. The School operates a grid across a number of laboratories.

5.4 Library and ICT support

A new library was built on the Talbot campus in 2003. Like its predecessor, it is octagonal - an interesting shape for a library. The library contains a number of workstations, although the image used supports general computing usage, and so does not include all the software required for the computing courses.

5.4.1 Network Performance

ICT Services have instituted a continuous virus checking of most files. This has had a significant impact on network performance, with compilation times frequently measured in tens of seconds, even for the simplest of source files.

5.5 Recreational

The campus has a small student village, but very little in the way of relaxing areas for students. There is a bar which serves food, and several other places to eat and obtain soft drinks.

As the campus is some 2 miles from the centre of Bournemouth, few students remain on the campus in the evening (apart from those studying or completing assignments).

6. My Background in Computing and Teaching

I started full-time teaching in 1974, 18 months after graduating. At that time it was a good financial move! Since then, my focus has varied somewhat (my masters is in Cognition, Computing and Psychology), but the continuing themes have been computing and industrial liaison. I am currently the Business Fellow (Computing), which means I visit companies on a regular basis, and initiate and support collaborative projects, such as Knowledge Transfer Partnerships.

At Bournemouth, the nature of my teaching load has changed considerably, due to the need to plug gaps as staff with programming expertise have left, and those with different skills have been recruited. In the past 15 years I have taught units in AI, HCI (design and programming), web development, OO programming, and eBusiness programming across a total of 5 undergraduate and postgraduate programmes.

I was the Programme Leader for the BSc Computing/Software Engineering combined programmes from 2002-2005. In Autumn 2005 I took over as unit leader for the Programming unit, the first time I have delivered an introductory programming unit since 1994, when I taught Ada on a conversion masters course.

6.1 Beliefs About Programming

The following beliefs about programming shape the design, delivery and assessment of any programming unit which I lead:

1. Programming is as hard as learning to read or write. It requires people to produce abstractions of their mental processes and represent these in an unforgiving, arcane format.
2. Unlike when students learnt to read and write, university students have an awareness of the progress of others, which can affect their learning, in both positive and negative ways.
3. Programming takes at least 5 years to tame, and 10 years to master.
4. The first experience of programming has a disproportionate effect on the development of the mental processes of programming. This means that the use of simple programming systems for introductory programming can impede understanding of the more fundamental concepts.
5. Comprehension is as important a cognitive skill, as algorithm design.
6. Discipline (in the form of software engineering principles) in the approach to application design, implementation, and testing is central to effective software development.
7. Optimising algorithms to minimise the use of computer resources, is no longer a high priority.
8. Imperative object-orientation is the dominant programming paradigm.
9. Object-oriented design is hard. Experts in any field tend to use object-oriented techniques, whereas novices tend to feel more comfortable with procedural techniques.
10. Object-orientation is not hard. In a procedural world, objects provide convenient mechanisms for contextualising nouns, verbs and adjectives.
11. Programming subsumes the language and the system. The size of the system is a significant and complicating factor in learning to program.

7. The Computing Undergraduate Framework

The first technical computing degree at Bournemouth University was the BSc Software Engineering Management (SEM), the first intake of which was 15 in the Autumn of 1989. A BSc Computing was added in 1995, which shared a common first 2 years with SEM until a review in 2001. A common first year was retained, and all but one unit of the second year was the same.

The first year student intake built from the 15 in 1989 to 100+ in 2000-2002. Since then, numbers have been restricted, firstly by staffing limitations, and latterly by reductions in applications.

The Computing Undergraduate Framework was developed in 2004-5, as part of the periodic review process, with a first intake in Autumn 2005. The size of the intake was 68, a big improvement over the 51 recruited to the Computing/Software Engineering Management combined programme in 2004. Applications for 2006 are 10% up on the previous year.

The framework consists of 4 programmes, all of which share a common initial 2 years. The placement is optional, although 'highly recommended'. The new programmes are Software Engineering, and Software Product Design. The core final year units for each of these programmes will be elective units for the others.

Throughout, the programmes have received BCS approval. In March 2006, the BCS conducted its periodic review, and granted full accreditation to each programme.

8. The Programming Unit

Other units in the first year include elements of programming. There is a Web Application Development unit, and another entitled Analysis, Design & Testing.

The other units are: Computing in Context, Systems & Networking, and Relational Databases. All six units carry 20 credits.

8.1 Paradigm

The paradigm of choice for the unit is object-orientation. This was not always so. The first programming language used on the SEM programme was Ada. This was briefly superseded by C/C++, and then there were three failed attempts with Java. C was reinstated as the first programming language in 2000. This remained the case until the Autumn of 2005.

8.1.1 Why Java?

The unit specification does not specify a particular language. However, there is an external constraint that affects the language choice: until Autumn 2005, the placement year between the second and final years has been mandatory. As students apply for placements during their second year, views of employers are taken into account. The feedback we have received is that employers are not keen to send placement students on *ab initio* training courses in programming languages. Therefore, the programme team have accepted that it is vital that the students have skills in programming languages which employers are familiar with. Whilst the placement year is now

optional, it is highly recommended, and applications from students wishing to complete a placement are looked upon more favourably.

Java has been selected over C#, due to its availability on multiple platforms. There are no immediate plans to change this. The School has an MSDNAA licence, and students are able to download and install most of the Microsoft software (apart from Office) free of charge. Many students avail themselves of this, and acquire VS skills in their own time.

8.2 The Role of the Unit in the Framework

The responsibility of the unit is to introduce the main elements of programming. This includes some design and testing, although these are covered in more detail in the Analysis, Design & Testing unit.

The programming units in the second year are Object-Oriented Programming, and Entertainment Systems, which focuses on multimedia programming.

8.3 My immediate background in teaching programming

I am not a believer in the use of languages like C as a first programming language. Previously, I had been the unit leader on the programming unit in the second year, and saw a very wide range of abilities, as one would expect. The issue was the inability of quite capable programmers to grasp the concepts of object-orientation, once they had a year of inculcation in the procedural paradigm.

However, there was little experience in the staff (including me) for an objects-first approach, so the existing C-based teaching scheme was modified for this first delivery.

8.4 Unit Specification

The Unit Specification is included in Appendix B.

8.5 Unit Specification Commentary

Each year I find that half of the students express dissatisfaction (verbally) with the recommended book. This year we are recommending Lewis and Loftus. This is reviewed annually.

The Library staff maintain booklists for the students, and use these as the basis for acquisitions. As one would expect, additional requests from members of staff are also accepted.

8.6 VLE

All units on the first year of the Computing Undergraduate Framework are using a slightly restricted version of the Blackboard VLE. The full, updated version will be available from Autumn 2007.

The VLE is being used as a repository for learning and assessment documents. Online submission and assessment (through Blackboard) are not currently being used.

In addition to lecture slides, worksheets, and assignment specifications, the materials include a number of tutorials. In previous years, I developed a simple website for my

materials. This has been integrated into Blackboard by using redirection. Currently, my website for this unit is available external to Blackboard at: (http://dec.bournemouth.ac.uk/staff/mjones/Teaching/unitIndex.php?unit=Computing-C-Programming).

8.7 Support software

We have tried IDEs in the past, and found the time taken for the students to become familiar with the IDE has been a significant problem.

The preferred editing/compilation combination is TextPad and command line Java compilation. TextPad provides Java compilation and simple execution facilities.

All students are 'strongly encouraged' to install the JDK and TextPad on their own computers. Nearly all the students have desktop and/or laptop computers, although ownership of same is not mandatory.

We using the version of Java current when the image is revised each July. For 2005/06, this is version 1.5.0_03.

9. Pedagogy

The earlier experience of using Java as a first language led to the publication of a few papers, but had disastrous results in terms of student learning and progression. It became clear that the concept of an object is far from intuitive. Rather than persevere, the decision was taken to re-trench to C, which has left a legacy of distrust regarding OO and Java.

The underpinning pedagogy for the delivery of this unit was to cover the fundamentals of programming and program design. Specifically, this meant that we intended to inculcate the students with the notion of a version of the 'iterate' design pattern. Rather than abstract the iteration, the aim was to expose the students to the low-level iteration through arrays, files and collections.

9.1 When to introduce Objects?

The decision was taken to introduce the use of objects early, but the design of new classes would be covered about halfway through the unit. This was reviewed at the end of the unit, and a more objects-first approach will be taken.

10.Instructional Design

The standard for units at the University is one (one hour) lecture and one (one hour) workshop or seminar per week. Programming is an exception, in that there are 3 contiguous hours of workshop per week, two of which are supervised.

Due to the timetabling system, there is no control over the proximity of the sessions. This year, for example, the lecture was on Friday afternoon, and the workshops on Thursday and Friday mornings.

The purpose behind the design of the unit was consolidation. The approach taken prior to 2000 was anarchic, with seven changes of direction in teaching approach and/or programming language in 10 years. The approach taken since that time has

yielded positive results, and the team took the view that radical redesign would be foolhardy.

10.1 Lectures

Currently, I use 2 types of lectures - the (largely) interactive and the (largely) presentational. Presentational lectures progress through a sequence of slides. These are of limited effectiveness in the context of programming.

Interactive lectures include the use of the whiteboard for written notes, and the computer to demonstrate specific techniques. The advantage is that students can see a solution evolving. There are drawbacks, especially as my writing is very poor. Also, these sessions tend to be less structured. This is something I will review this summer.

Interweaving presentation slides and interactive periods has proven to be disruptive to students. A better solution would be two data projectors per lecture theatre. The obvious solution of an overhead projector and a data projector is not possible, as they both use the same screen in each lecture theatre.

10.1.1 Interaction

I ask questions, but there were two problems with this cohort. In addition to the traditional problem of a small coterie of students answering nearly all the time, the lecture theatre allocated this year was wide and shallow, making it difficult to interact with a significant percentage of the students at any given time.

10.1.2 Feedback

Currently, I do not use any specific feedback mechanisms for the lectures. I will consider some of the options identified by other members of the Disciplinary Commons.

10.2 Programming Laboratory sessions

Each group of 18 students has a 3 hour programming workshop each week. The first and last hour is supervised by one member of staff (not necessarily the same person).

10.2.1 Weekly worksheets

Each student is provided with a weekly worksheet (available via the VLE) (see Appendix D). These include a number of exercises (typically 8-12) most of which involve constructing applications. The remainder focus on adding functionality to an existing application.

Students work through the exercises at their own pace, asking questions where necessary. Staff are encouraged to circulate, providing assistance and questioning students on their progress. All feedback is verbal.

Students were encouraged to reflect on their progress via a journal, but this was not highly structured, and few participated.

10.2.2 Multiple choice

The original idea was to make a multiple choice test available for each workshop, but this proved too large a development load. A simple web-based delivery application

was written, and a few tests made available. The results of these tests were not recorded, and students could continue answering the same question until they selected the correct answer. (Appendix E contains a link to lesson tests).

A version of this was made available for revision purposes.

A secure version was used in the summative assessments.

11.Assessment

The assessment regime is, inevitably, a compromise between one's beliefs concerning assessment, and practical considerations regarding student workload on this and other units.

11.1 Assessment regime

The unit is assessed via a combination of coursework (50%) and unseen, end of unit examination (50%).

11.1.1 Rationale for Examination

For the past 10 years there has been no examination in first year Programming. The corresponding second year unit does have an examination, with the effect that the first experience that students have had of an examination in programming was at a stage when the results contribute to the final award. The decision was made to have an examination in each of the programming units.

Until the students complete the examination, one cannot know how this will go. There is a certain amount of apprehension in the teaching team.

11.1.2 Coursework

The coursework consists of 4 separate pieces of work:

a. A multiple choice online test, and a short written test
b. An electronic logbook of 5 applications
c. Another multiple choice online test, and a short written test
d. Construction of one of 3 applications

This follows the format of previous years - another facet which will be revisited at the end of this academic year.

11.2 5 Beliefs About (Summative) Assessment

My beliefs about assessment are:

- Assessments should be multi-level, reflecting the range of student capability
  - students should be able to choose the level they wish to attempt
- Each assessment should involve new knowledge
- Plagiarism should be discouraged
- If the assessment involves writing a program, then the program should compile and execute
- Marking:
  - Students should see where marks:
    - will be allocated

        - have been allocated, and why
    - It should be easy to mark

11.3 Individual or Group?

All the assessments in the first year are individual. Again, this is a tradition on first year programming. This will be reviewed at the end of this academic year.

11.4 Formative assessment

Students are supplied with a number of exercises each week (typically 8-12), and multiple choice 'lesson tests'. They are provided with verbal feedback in the context of the workshop. Formative assessment is one area we hope to increase in the next delivery. The other is evidence of reflection, in the form of journals. In this delivery, journals are optional, and have had a low take-up.

Students are also encouraged to email their completed exercises, for comment. Few do this, but they receive written feedback.

11.4.1 Lesson Tests

In the early weeks, students have access to a couple of online tests, which allow them to see the answers. These are available at:
http://dec.bournemouth.ac.uk/staff/mjones/Teaching/runTests.php?unit=Computing-C-Programming

Unlike the summative multiple choice tests, these allow students to continue selecting until they get the right answer.

In the next delivery (this is the first with the current delivery team), more tests will be available.

Students do not receive written feedback on non-assessed work. Partly, this is because the marking of the assessed work takes so long. Simpler-to-mark summative assessments will (hopefully) facilitate the availability of formative feedback.

11.5 Sample Summative assessment

This is included in Appendix G.

One of the pertinent issues relating to this assignment is the timing of the assignment. Assignment 1 was submitted at the end of the Autumn term, some 4 months before the due date of this assignment. In this time gap students focused on assignments in other units, thus limiting the 'continuous' nature of the assessments in Programming.

11.5.1 How the beliefs realised in the sample assignment?

1. Multiple applications are specified, each with a different level of difficulty. Students can select which they attempt. As the marking scheme requires that the application functions (at some level), students are encouraged to attempt the application which most suits their abilities.
2. The new knowledge involved involves recursively descending through a file system.

3. Students are required to demonstrate their applications, and explain certain elements (see Appendix H for the demonstration checklist). Explanation and comprehension cannot, in themselves, detect plagiarism, but they do ensure that those receiving 'help' must play some role in the code production.
4. The simplest application, the copier, should be achievable by every student, as previous exercises have involved both reading and writing files. Getting an application to compile and execute is an essential part of programming.
5. The focus in the marking is on the application executing and the documentation being in place. These are relatively easy to mark. Issues such as the efficiency of the algorithm are briefly covered in the feedback.

11.6 Providing Feedback

Appendix H contains links to examples of oral and written feedback.

Informal and formal feedback is provided. The informal, verbal feedback is very informal – the emphasis is on minimising weaknesses, in order to limit the loss of motivation, especially for the weaker students. Formal feedback is provided in written form, which was augmented in the first assignment of the year by oral feedback.

11.6.1 Written

The online multiple choice tests produce two forms of output - an immediate mark, showing the total number of questions answered correctly. A complete listing of questions, answers, and correct answers is made available after a week or so (when all have completed the test).

Written comments are also made on the other assignments. Some of this is auto-generated, with adjectives used for excellent to indifferent performance. Additional comments are usually made.

11.6.2 Oral

This year, I used oral feedback, by recording my comments as I looked through the code. The reaction amongst the students to this was not measured (it should have been), but anecdotally some of the students felt that it was useful.

One difficulty arose - the default recorder supplied with Windows only allows for 60 seconds of recording from a given point. For longer recordings, one has to stop the recording, and then start again. One tends to watch the clock whilst recording. An improved mechanism will be used next year.

11.6.3 Availability of Feedback

Students receive their feedback electronically, via a web page which requires a login.

12.Feedback on the Performance of the Unit

12.1 University mechanisms

The University operates a system which, until last year, was called TLAS (Teaching and Learning Assessment Survey). It is now called AUE (Annual Unit Evaluation) (See Appendix J). The idea is the same - Lickert-scale responses required from a dozen or so questions on each unit (as well as on the facilities, etc.)

The survey is distributed towards the end of term 2, well before the final assignment or examination. The value of the responses is therefore questionable, and are used only for general guidance.

Students write their answers, although there are plans to put it online. A previous experiment in that direction allowed multiple (anonymous) entries by a single student, so proved worthless.

Each programme produces a Unit Monitoring Report (Appendix I), which includes analysis of student feedback and student performance, as well as reflections by the unit leader on the conduct of the unit, and plans for the next presentation.

12.2 Programme mechanisms

In order to facilitate more timely feedback, the programme has introduced two mechanisms.

12.2.1 Intermediate questionnaire

A TLAS-like survey was conducted before Christmas, so that the results can then be fed into the delivery of the rest of the unit. This was the first year of the use of this process. The Programming unit received very positive responses.

12.2.2 Course tutorials

For the past 10 years, there have been scheduled sessions each week for students (in their seminar groups) to provide verbal feedback to a member of the lecturing staff. If there is a pattern to the comments, this is fed back to the unit leader.

More serious or persistent problems are reported through the formal Programme Management meetings (as are successes).

13.Reflections

These are my personal reflections on various aspects of the design and delivery of the Programming unit for the academic year 2005-06.

13.1 The Environment

13.1.1 University

The University is a changing environment - this has been the pattern of the past 20 years, and is set to continue. The new focus on research is welcoming, but the pace of change is challenging.

13.1.2 School

Computing is the last element in the name of the School, and has suffered a worse staff student ratio than other parts of the School and University. This has been mainly due to the failures and delay in replacing staff who have left.

13.1.3 Programme

The explicit aim of the programme is to smooth the path of students into well paid employment in the Computing industry. This vocational emphasis has exerted pressure to 'chase the technology', which can deflect from improving the delivery of the units.

The programme has also had a strong scholarly element, and this, combined with the vocational elements, has led the programme to have the reputation across the University as the hardest undergraduate course.

13.1.4 The University Experience for Students

The pressure to succeed, from parents and bank managers, as well as from the students themselves, has led to a noticeable increase in tension amongst students. Add to this the high demands of the programme, and one generally finds that Computing students are the most stressed in this University.

The limited relaxation facilities and available programming laboratories only serve to reduce the satisfaction element of the student experience. Bournemouth students, and Computing students in particular, have made highly negative returns to the National Student Survey.

The picture is not all gloom, with most students going on to highly successful careers. Their student experience, though, is less pleasurable than it could be.

13.2 The Unit

13.2.1 Overall

As this is the first cohort on the revised scheme, and the year is not complete, it is not wise to give an overall verdict. Until the start of term, I would have said that things were going well, but there were a number of non-submissions for the second assignment.

13.2.2 What has gone well?

The weekly worksheets have provided students with a framework to develop their skills. The number of exercises also permit a level of privacy, where students can appear to be working on different exercises, thus reducing the ability for comparisons. The negative side of this is that students can hide.

The traditional (on the course) negative views concerning programming have not manifested themselves, and the attrition rate has been lower than usual.

The assignments were set on time, and the marking completed as per the University stated requirements (within 3 weeks). All materials were available by the scheduled sessions.

13.2.3 What has gone badly?

The levels of feedback are too low. This leaves the students somewhat lost, and the delivery team uncertain of where to focus their energies.

13.2.4 Did the Delivery Plan Work?

The main elements of programming were introduced in a reasonably logical order, which permitted students to gain appropriate skills. In reviewing the plan, three pedagogic problems were identified: the emphasis on numbers, the reliance on time-based arrays, and the late introduction of classes.

Numbers

The previous C-based delivery programme had focused on numerical examples. In itself this was not a problem, but it delayed the introduction of an objects-based focus. More work earlier in strings would have been more beneficial.

Time-based Arrays

As arrays were not introduced until the Spring term, the algorithms in the first term were restricted to the 'read a number in a loop' variety. This meant that the intention to cover iteration through datasets initially focused on files. Arrays were then the focus, followed by collections, using the file system as an example of a recursive dataset.

Although the focus on files was partially successful, the sequencing was awkward. A number of weeks were spent on basic loops, but all the examples used files (keyboard, screen and disk-based). Without arrays or collections, the examples used what I term 'time-based' arrays, where the same scalar variable is used in each iteration to hold a different value. I believe this is difficult for the students to comprehend, as it involves them understanding the concept of scoping, long before we have formally covered the subject.

Late Introduction of Classes

The focus on procedural aspects in the initial stages was intended to be ameliorated by the use of an application design from the outset which included separate 'App' and 'business' classes. This was not wholly successful, as a number of students later assumed that no logic could be placed in 'main', other than to instantiate a business object.

Objects were introduced in a period between assignments. This caused a problem, as students were focussing on assignments in other units.

13.2.5 What is Excellent about the delivery?

Basically, nothing. The various elements of materials, assessment, and feedback are present, but there is little evidence of practice which could usefully be passed on to others.

13.2.6 What innovations are there?

Again, it was essential to build a platform which can be developed, and into which innovative ideas can be introduced. This has been achieved.

The minor innovations were the use of recorded oral feedback for assignments, and the generation of feedback documents from spreadsheet marks and comments files.

13.2.7 What changes will be made?

The ordering of material will be changed, with a greater emphasis earlier on constructing loops. The use of collections and arrays will also feature earlier. The rationale for this is that the primary pattern for the first year is iteration through dataset, and the delaying of arrays until term 2 compromised this, to an extent.

The main organisational change will be to increase information flow, starting in the induction week. Students will be canvassed as to their previous programming experience. This may be used to influence the selection of seminar groups, to include some element of streaming.

Feedback on student progress will focus on each of the components of programming.

Student feedback on the progress of the unit will be facilitated by the wider use of journals, as used by Pete Bibby. These will be integrated into the formative assessment.

Finally, the timetabling of the lectures and workshops will be harmonised, with the idea of a 'programming day', where all the workshop sessions will take place.

13.3 My Performance

In terms of the mechanics, the materials and assessments were in place when required. Feedback to students did not attract any negative comments, and the performance of a number of students was outstanding.

There was a degree of tentativeness in my performance in this unit, which surprised me. I was much less confident of delivering introductory programming than I had imagined I would be.

14.Final Thoughts

As software becomes more pervasive and more sophisticated, the requirement for programming will increase, to provide the necessary contextualisation that a generic piece of software cannot provide. The nature of this programming will vary as it does not, from configuration files, through spreadsheets to programming languages. The acquisition of some level of programming capability will become more important. The teaching of programming will, in future, face ever greater challenges.

Appendix A: The University

Entrance

Poole House

Atrium

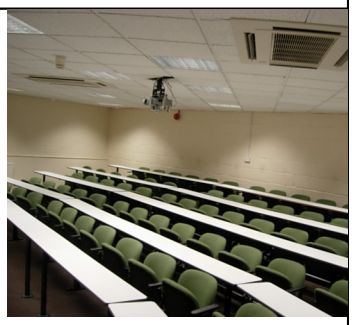
Cafe

Mission Statement

Library Cafe

Workstation

Library & Sculpture



Stevenson Lecture Theatre

Lots of Rocks

Courtyard

Library

Appendix B: Unit Descriptor

Unit number

Unit title **PROGRAMMING**

Level **C**

Credit value **20**

PRE- AND CO-REQUISITES

None.

AIMS

The role of this unit in the programme is to provide the learner with the fundamental skills and understanding of algorithm construction in modern programming languages.

The aim is to develop competence in the learner, so that they may write well designed, well structured, and adequately tested programs that meet given specifications, using a single programming language.

The unit concentrates on the Programming Fundamentals aspect of the Computing benchmark and also includes topics from Data Structures and Algorithms.

INTENDED LEARNING OUTCOMES

At the end of the unit the learner is expected to be able to:

1. Design and implement algorithms to solve simple, well-specified problems, using sequence, selection and iteration.
2. Demonstrate the appropriate use of data structures in simple programs.
3. Reason about the correctness of algorithms using pre- and post- conditions.
4. Execute functional and structural tests of simple programs.
5. Demonstrate the appropriate use of documentation techniques and tool support to communicate to others the design of the programs he/she has developed.
6. Comprehend and construct programs written in a programming language which supports object-orientation.

LEARNING AND TEACHING METHODS

Background

This unit is the focus for the development of skills in, and the understanding of, the design, implementation and testing of small computer programs.

The treatment is to teach programming from first principles in a disciplined and design-focussed manner. The unit aims to concentrate on the principles of good software development, rather than simply teach the syntax of any given language. Hence, there should be useful lessons learned even for those already familiar with programming.

The approach taken in the unit is system building 'in the small', enabling the solutions to a variety of problems to be implemented. Problems studied will bear some relationship to real-world problems, but be simplified to enable concentration on the techniques and disciplines involved. The presentation of

the unit will stress reasoning about problems, and design, before the production of program code. Example problems and their solutions will be used to demonstrate the techniques involved and the importance of structured testing and appropriate documentation will be stressed.

Indicative Styles

The unit will be delivered through a mix of lectures and practical workshops (labs). The lectures will develop programming concepts, while the students will use the workshop time for their practical software development work. Each lecture topic is matched with a set of graded lab exercises, which enable the learner to put the concepts from the lecture into practice.

Feedback

Learners will be given direct feedback on their example programs by supporting staff in labs. In addition, they will be given written feedback on programming assignments. Written tests will be discussed, teaching staff pointing out particular strengths and weaknesses.

ASSESSMENT

Assessment Weighting

The weighting of coursework to examination is: 50:50

Assessment Regime

All ILOs will be assessed through coursework and/or examination.

A typical assessment regime will be:

ILOs 1-6: assessed by practical programming assignments

ILOs 3 and 6: assessed through written tests

ILOs 3 and 6 and (to some extent) 1 and 2: assessed through an end-of-unit unseen examination.

INDICATIVE CONTENT

Elements of computer programs
Variables and types of data. Executable statements. Sequence, selection and iteration.

Data Types
Simple data types: scalars, references. Aggregate data types: arrays.

Algorithm design
Design decisions. Loop design. Compound conditions. Recursion. Error and exception handling.

Program aggregation
Simple classes, using aggregation. Methods. Value and reference parameters. Instance and class data. Library classes. Comprehension of single inheritance.

File and Directory processing
Input and output of files, streams, and directories.

Program quality
Code layout, naming conventions, comments. Correctness - pre- and post-conditions. Structural and functional testing. Test coverage. Test case selection. Debugging.

INDICATIVE KEY LEARNING RESOURCES

Books

John Lewis, William Loftus (2004), *Java Software Solutions (4th Ed.).* Addison-Wesley.

Kathy Sierra, Bert Bates (2003). *Head First Java,* O'Reilly & Associates.

Steve McConnell (2004). *Code Complete: A Practical Handbook of Software Construction (2nd Ed.).* Microsoft Press International.

Journals

Communications of the ACM

Journal of Object-Oriented Programming

ACM SIGCSE

ACM SIGPLAN

Web-based sources

jGrasp: http://www.jgrasp.org/

Support website for Lewis & Loftus: http://duke.csc.villanova.edu/jss1/

Unit support page: http://dec.bournemouth.ac.uk/support/comp/prog/

Appendix C: Unit (Delivery) Guide

	Lecture Date (Fri)	Lectures	Workshops	ILOs	Directed Reading
1	07-Oct	Java: Hello World (SW)	Java: Hello World	1	Lewis & Loftus: Chapter 1 - pages 26-42
2	14-Oct	Java: Variables and pre-defined arrays (SW)	Java: Variables and pre-defined arrays	1,2	Lewis & Loftus: Chapter 2 - pages 61-84
3	21-Oct	Java: Writing Text Files (MJ)	Java: Writing Text Files	1,2	Lewis & Loftus: Chapter 2 - pages 88-92
4	28-Oct	Java: Reading Text Files (MJ)	Java: Reading Text Files	1,2	Lewis & Loftus: Chapter 2 - pages 88-92
5	04-Nov	Java: Repetition using fixed count loops (MJ)	Java: Repetition using fixed count loops	1	Lewis & Loftus: Chapter 5 - pages 245-251
6	11-Nov	Java: Selection (SW)	Java: Selection	1	Lewis & Loftus: Chapter 5 - pages 201-226
7	18-Nov	Java: Repetition using variable count loops (SW)	Java: Repetition using variable count loops	1	Lewis & Loftus: Chapter 5 - pages 227-237
8	25-Nov	Programming: Algorithm design (MJ)	Programming: Algorithm design	1,3	
9	02-Dec	Programming: Introduction to testing (SW)	Programming: Introduction to testing	4,5	
10	09-Dec	Programming: Testing strategies and creating data (SW)	Programming: Testing strategies and creating data	4	Lewis & Loftus: Chapter 3; 124-126
		Christmas Vacation			
11	13-Jan	Java: Creating and processing arrays (SW)	Java: Creating and processing arrays	1,2	Lewis & Loftus: Chapter 7 - pages 369-379
12	20-Jan	Java: Searching and sorting arrays (SW)	Java: Searching and sorting arrays	1,2	
13	27-Jan	Java: Writing methods (MJ)	Java: Writing methods	1,6	Lewis & Loftus: Chapter 6 - pages 319-327
14	03-Feb	Java: Program blocks; variable and method scoping (MJ)	Java: Program blocks; variable and method scoping	1,2	Lewis & Loftus: Chapter 6 - pages 319-327
15	10-Feb	Java: Sharing data between methods (SW)	Java: Sharing data between methods		
16	17-Feb	Project Week	Project Week		
17	24-Feb	OO: Introduction (MJ)	OO: Introduction	1,6	Lewis & Loftus: Chapter 6 - pages 287-308
18	03-Mar	Java: The final and static qualifiers (MJ)	Java: The final and static qualifiers	1,2	Lewis & Loftus: Chapter 6 - pages 291-295
19	10-Mar	Programming: Application	Programming:	1,5,	

		architectures (SW)	Application architectures	6	
20	17-Mar	Java: Introduction to recursion (MJ)	Java: Introduction to recursion	1	Lewis & Loftus: Chapter 11 - pages 575-588
21	24-Mar	Java: Introduction to collections (SW)	Java: Introduction to collections	1,2, 3	Lewis & Loftus: Chapter 12 - pages 611-620
		Easter Vacation			
22	21-Apr	Java: More collections: hash tables (SW)	Java: More collections: hash tables	1,2	
23	28-Apr	Java: Revision (SW)	Java: Revision	1,2	
24	05-May	Programming: Revision (MJ)	Programming: Revision	1,2, 6	
25	12-May	Revision	Revision	3,4, 5	

Appendix D: Weekly Worksheet Example

Computing Undergraduate Framework

Programming Workshop Week 4

Java – Reading Text Files Exercises

1. Compile and run the ReadFile application. Run the application, and input the values requested.

2. Create a text file using TextPad, and then use the input redirect command to take the input from this file, not from the keyboard. Comment on the output.

3. Create an application which takes the name of the text file to be read from the command line, and then displays the first 2 lines of that file.

   An example command line would be:

   java ReadFileApp "h:\MyTextFile.txt"

   (You only need to enclose the filename in double quotes if it contains spaces.)

4. Repeat the previous command, this time using the Unix directory character '/' instead of '\'. Does it work?

5. Write an application which reads 3 numbers from the user, and adds them up, and prints out the result. Each number is to be stored in a different variable.

6. Run the previous application, this time taking the input from a file using input redirection.

7. Modify the application to read the input from a text file (not using input redirection).

8. Write an application which produces the sum of 4 numbers entered by the user on the command line. For example:

   java SumCommandLineApp 3 2 1 4

   Will produce: The sum of the 4 numbers is 10.

   You will need to convert the command line words (which are stored as strings) into integers. You will find the 'parseInt' method of the Integer class useful.

9. Modify the previous application to allow the user to enter non-integer numbers. Look at the definition of the 'Double' class.

10. Write an application which has the following behaviour: the user specifies the names of three input files on the command line. The

application is to read the first line of each of these files, and then write them (in order) to an output file.

Appendix E: Sample multiple choice questions

Link to Lesson tests

Appendix F: Sample Summative Assignment

School of Design, Engineering & Computing

Course Computing Undergraduate Framework
Year Level C
Unit Programming
Assignment Two

Issue Date: 14 March, 2006
Due Date: 25 April, 2006

Introduction

This assignment addresses three of the intended learning outcomes of this unit, specifically –

4. Execute functional and structural tests of simple programs.
5. Demonstrate the appropriate use documentation techniques and tool support to communicate to others the design of the programs he/she has developed.
6. Comprehend and construct programs written in a programming language which supports object-orientation.

Individuals and Groups

This is an individual assignment, but students can work in groups of three.

There are 3 applications. Each student elects which application to attempt. As the output from application 1 is the input to application 2, and so on, students can work in groups to produce an application pipeline. This is optional.

Note that the applications are not equally difficult. It is important that the application you write actually works. Therefore, choose the application which most closely suits your ability.

NOTE: it is important that you follow the instructions for formatting your code. More details, plus an example, are available from the tutorial available at: http://dec.bournemouth.ac.uk/staff/mjones/Teaching/tutorials.php?unit=Computing-C-Programming

The Concepts

Each application covers a number of concepts, each of which will result in some code. It is the responsibility of the student to annotate his or her code, so the coverage of the concepts can be highlighted. See the tutorial available at the address above.

Failure to correctly highlight the concepts will result in a significant loss of marks.

The Applications

Application 1: The Creator Difficulty: Challenging
This application creates a directory tree, containing files with various extensions. The root directory will be the name of the group (e.g., Knuth). At each level, a random number of files (between 0 and 30) and a random number of directories (between 0 and 5) are to be created. The maximum depth of directories is to be supplied on the command line of the application.

An example command line will be:

java CreatorApp Knuth 5

Indicating a maximum depth of 5 levels, including the root directory.

All filenames must start with the string 'file_', and directories with 'dir_'. The rest of the name (apart from the extension) must be of the format aaaa_nnnn_aaaa_nnnn, where 'a' is a lowercase letter, and n is one of the digits (0-9).

Each file will have one of the following extensions: txt, java, php, html. The contents of each file will be relevant to the extension (i.e., the Java file should compile correctly), and must contain (in a comment at the start of the file): the pathname of the file (including the root directory), the filename, and the date and time which it was created (using the YYYY-MM-DD-HH-MM-SS format).

I.e., in a PHP file:

```
<?php
          // Path: Knuth/
          // File: file_aaaa_0000_bbbb_1111.php
          // Date created: 2006-04-01-12-15-00
          // Date changed: 2005-10-02-52-35-00
```

The last changed (modified) time of each file should be set to a random time within (approximately) the past 12 months.

The file should also contain a random (between 1 and N – the number of words) subset of words from a file called words.txt (which is in the same

directory as the creator application). These should be embedded in the file, for example in strings, or as variable names or contents of HTML tags.

The application should also log each file which is created, in a file called 'filesCreated.txt'. Each line of this file will contain the pathname of the file, the last modified time which has been set, and the subset of words which have been selected.

The filesCreated text file should be placed in the root of the created directory tree (Knuth, in this case).

NOTE: created directories should be indicated by a trailing '/'. (Forward slash should be used as a directory separator).

NOTE: all pathnames are taken from the root directory of the group, not of the file system. So, if you are in the 'Knuth' group, all your pathnames will start with 'Knuth/' The first item created will be the Knuth directory itself, so the log file (filesCreated.txt) will contain the first line:

```
Knuth/ 2006-04-03-12-13-56
```

Indicating that the directory was created at nearly 12:14 on the 3rd of April, 2006. An example for a file would be:

```
Knuth/file_aaaa_0000_bbbb_1111.html     2006-04-03-12-13-
56 2005-11-02-10-30-00 first king
```

The line shows that the file was created at the same time as the directory; last modified time of the file was nearly 25 past 3 on the 13th of September 2005, and that 'first' and 'king' were selected from the set of words in the words.txt file.

NOTE: a word will be taken to be any sequence of letters (A-Z, a-z).

Concepts Involved

The Java concepts included in this application are:

1. class
2. attribute
3. constructor method
4. non-constructor method
5. method signature
6. writing to a file
7. reading from a file
8. checking the command line
9. creating a directory
10. using a collection
11. handling an exception
12. randomly selecting from a collection
13. accessing file characteristics (e.g., modified time)

14. using a Java package

NOTE: an application may contain multiple instances of the use of a concept.

Application 2: The Crawler Difficulty: Hard

This application identifies files which could potentially be archived and copied. A file is to be archived if the last modified date is more than 100 days ago. A file is to be copied if it has been modified since a given time. Only nominated extensions are to be considered. The java.util.Calendar class contains relevant methods for comparing times and dates.

For instance, a command line of: java CrawlerApp Knuth 2005-12-28 php html

Will nominate for archiving or copying only php or html files in the Knuth directory tree. It will only nominate for copying files modified after midnight on the 28th of December, 2005.

The output from this application is two files (called archive.txt and copy.txt) containing the names of files to be archived, and those which may be copied. The names will follow the convention specified in the create application. These two text files must appear in the root directory (Knuth, in this example).

A simpler version of this application will archive none of the files, and identify for copying all files with the nominated extensions.

NOTE: a word will be taken to be any sequence of letters (A-Z, a-z).

NOTE: this application is NOT to read the log file created by the create application. Any application found to be doing this will be awarded a mark of zero.

Concepts Involved

The Java concepts included in this application are:

1. class
2. attribute
3. constructor method
4. non-constructor method
5. method signature
6. writing to a file
7. reading from a file
8. handling an exception
9. checking the command line
10. string manipulation
11. accessing file characteristics (e.g., modified time)
12. using a Java package

NOTE: an application may contain multiple instances of the use of a concept.

Application 3: The Copier Difficulty: Medium

This application reads the archive.txt file, and archives (copies) the files listed in it to a directory called ‘archive’ in the root directory (Knuth in the previous two applications). All files will be in the same directory, with the name of the file reflecting its location. All ‘/’ in the path will be replaced by ‘_slash_’. So, a file called 'Knuth/file_abcd_0123_efgh_4567.php' will be archive to a file called: 'Knuth/archive/Knuth_slash_file_abcd_0123_efgh_4567.php'

NOTE: you can assume that all files to be copied are text files.
NOTE: archived files are NOT to be deleted.

A similar process will be followed for copying files, except that the destination directory is to be called ‘copy’, and that files will only be copied if they contain at least one of a set of specified words. These words are to be specified on the command line. If no words are supplied, all files are copied. The filenames are to be modified as per the archive, with ‘/’ being replaced by ‘_slash_’.

An example command line is:

Java Copier fred Bloggs ate my hamster

Will archive all files identified in the archive.txt file, and copy all files listed in the copy.txt file which contain ANY of the words on the command line (from 'fred' onwards).

Concepts Involved

The Java concepts included in this application are:

1. class
2. attribute
3. constructor method
4. non-constructor method
5. method signature
6. writing to a file
7. reading from a file
8. handling an exception
9. using a collection
10. using a Java package

NOTE: an application may contain multiple instances of the use of a concept.

Submission
There will be one CD submission per student, which includes all the documents.

The structure of the CD must be:

Top level directory

File called 'Personal Details.txt', containing the names of the student and his or her group.

Directory called 'Application'
One directory for the application – called Creator, Crawler, or Copier
Source directory
Java files
Object directory
class files
JavaDoc directory
Documentation (JavaDoc) files
Annotated directory
A single annotated file

Directory called ‘Tests’
One directory for the application – called Creator, Crawler, or Copier
Test plans and results (RTF or PDF files)
Batch file to compile and execute the application
One generated directory with the name of the group

Documentation
Each of the applications must be documented using the JavaDoc documentation guidelines.

The following are considered important:

a. appropriate choice of class, data, and method names
b. consistent indentation
c. appropriate use of comments within methods
d. appropriate use of JavaDoc comments for methods and classes

Note: the marking scheme will punish those who skimp on these areas.

Testing
You must provide evidence that each of the applications compiles and executes successfully.

Evidence will include:

a. Structural test plans and results analysis (in RTF or PDF format)
b. Captured output from the compilation and execution of the application
c. Input and output files (where relevant)

Demonstration
Students must demonstrate the compilation and execution of their applications in the relevant workshop in the week of the assignment submission.

Presentation of the Results
The output of the application should be presented as appealingly as possible - using, for instance, HTML and/or PHP. The more pleasing the output, the more marks.

Marking Scheme
The marks for each student will be allocated as follows: there will be a ‘raw’ mark, which is then modified (multiplied) by a 'difficulty' mark, and a ‘style’ mark.

Raw Mark

Category	Mark
Evidence of correct execution, in the context of a demonstration Compilation – 10%, execution – 15%	30%
Evidence of appropriate testing, including structural test plans + results	20%
Correct highlighting of the coverage of concepts - using HTML embedded in Java comments - see the tutorial	30%
Presentation of the results of the application	20%

Difficulty

Category	Mark Multiplier
Crawler	1
Creator	0.85
Copier	0.7

Style

Category	Mark Range
Evidence of documentation (including JavaDoc), code layout, appropriate use of variable names	0 - 1

Good style means appropriate use of:
- variable, class, and method names
- classes - i.e., more than one
- consistent indentation
- JavaDoc comments on methods and classes

Therefore, if you produce a working application, but don’t bother to document the code, and use variable names with single letters and/or starting with capital letters, and put everything in a single class, then you will be allocated a style mark close to 0. That will produce a low overall mark, whichever application you choose.

Signature Lecturer:

Signature QA:

Date:

Appendix G: Demonstration Checklist

Computing Undergraduate Framework

2005/06

Programming Assignment 2 - Demonstration

Date:

Student: **Group:**

Supervisor: **Mark:**

Application demonstrated: Creator Crawler Copier

Demonstration Files are available from:
http://dec.bournemouth.ac.uk/staff/mjones/Teaching/assignments.php?assignment=2&unit=Computing-C-Programming

The application compiles without errors: 0 / 1 / 2

The application executes and produces no errors: 0 / 1 / 2

The application produces the required output:

Creator: directory + filesCreated.txt + different last modified times 0 / 1 / 2 / 3 / 4
Copy the demonstration words.txt file
java <app> Berners-Lee

Crawler: archive.txt + copy.txt 0 / 1 / 2 / 3 / 4
Copy + unzip the example demonstration zip file
java <app> Berners-Lee 2006-01-24 html

Copier: files in archive.txt copied to archive + same for copy.txt 0 / 1 / 2 / 3 / 4
Copy + unzip the example demonstration zip file
Copy + unzip the demo archive and copy zip file
Copy the archive.txt and copy.txt to Berners-Lee
java <app> Berners-Lee java do

The student is able to identify and explain the purpose of FOUR concepts: 0 / 1 / 2 / 3 / 4

List: constructor / Open output file / loop / method

JavaDoc comments on methods, classes 0 / 1 / 2 / 3 / 4

The JavaDoc files are available: 0 / 1 / 2 / 3 / 4

Appendix H: Example of feedback given to students

Student written feedback

Student Oral Feedback

Appendix I: Unit Monitoring Form

UNIT MONITORING REPORT

Unit Title	Unit Reference	Academic Year
Programming		2005-06

Unit Leader	School	Partner Institution
Michael Jones	DEC	

Programme(s) in which unit is offered

Computing Undergraduate Framework

Key strengths and issues arising from student performance

Key strengths and issues in student feedback

Key strengths and issues arising from other unit monitoring data (including any specific references in an External Examiner's feedback)

UNIT MONITORING REPORT

Actions for improvement in next year / cycle of delivery

Other comments

Signature of Unit Leader _____ Date _____

Received by Programme Leader _____ Date _____

Appendix J: Student Feedback Questionnaire

Bournemouth University Annual Unit Evaluation Questionnaire

Please spend a few minutes completing this questionnaire. The results will be used to improve the quality of this unit. For each of the statements below please use blue or black ink and mark with a cross **X** the box which best indicates your view. No response will be interpreted as *not applicable*. We welcome all the information that you provide and assure you it is treated **anonymously**.

Letters *Numbers* *Letter*

Unit Code [ ][ ][ ] --- [ ][ ][ ][ ] --- [ ]

My attendance on this unit was:

1. 0-24% ☐ 25-49% ☐ 50-74% ☐ 75-100% ☐

(X in the boxes please)

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
The tutor(s)...					
2. organised the unit well.	☐	☐	☐	☐	☐
3. issued handouts and other materials which helped me to understand the unit.	☐	☐	☐	☐	☐
4. used a variety of learning methods which worked well.	☐	☐	☐	☐	☐
5. gave plenty of examples and illustrations which helped me to understand the unit.	☐	☐	☐	☐	☐
6. used the VLE to help me to understand the topics.	☐	☐	☐	☐	☐
7. shared their enthusiasm about the subject with us.	☐	☐	☐	☐	☐
8. could be contacted for advice if needed.	☐	☐	☐	☐	☐
9. made it clear to me what was expected in the assessed work for this unit.	☐	☐	☐	☐	☐
10. gave me the support I needed to help me complete the assessed work for this unit.	☐	☐	☐	☐	☐
11. made constructive comments on my assessed work.	☐	☐	☐	☐	☐
When I was studying this unit...					
12. I tried to make sense of things by linking them to what I knew already.	☐	☐	☐	☐	☐
13 I tried to relate ideas I came across to other topics wherever possible.	☐	☐	☐	☐	☐
14. I paid careful attention to any advice or feedback I was given.	☐	☐	☐	☐	☐
15. I prepared for classes in advance.	☐	☐	☐	☐	☐
16. I worked steadily during the course, rather than leaving things to the last minute.	☐	☐	☐	☐	☐
17. I was systematic and organised.	☐	☐	☐	☐	☐
18. I could see the relevance of what we were taught.	☐	☐	☐	☐	☐
19. I found it easy to work with other students.	☐	☐	☐	☐	☐
20. I checked my assessed work to ensure that it really met the requirements.	☐	☐	☐	☐	☐

Thank you very much for filling in the questionnaire.
Please use the back of this questionnaire if you wish to make any further comments.