Disciplinary Commons –
Portfolio on the course Introduction to Computing

Monika Seisenberger
University of Wales Swansea

Abstract

This is a portfolio on the course Introduction to Computing I, II taught at the University of Wales Swansea, in 2005/2006. The purpose of this portfolio is to understand the context of the course, to reflect on the way in which I have taught the course, and to establish a way of how it seems to be right for this course. One characteristics, in which this course differs from the others in the Disciplinary Commons, is that it is a course for Non-Computer-Scientists.

Contents

1 Context

1.1 What it is like to teach in Swansea

The Department of Computer Science (http://www.swan.ac.uk/compsci/index.html) at the University of Wales Swansea is a relatively small department with around 20 members of teaching staff. The main research disciplines are logic and computation, visual and interactive computing, scientific computation and future interaction technologies.

The intake of students, who are majoring in Computer Science, is around 100. Beside the BSc Computer Science scheme there are several joint honours schemes; the numbers of students on these schemes however are smaller. Swansea also offers an MSc course in Computing and Software Technology, which is very attractive for foreign students, and has recently started several MRes Programmes.

One reason for students to study in Swansea is its beautiful surrounding. Swansea is Wales' second largest city and is located in the centre of the South Wales coast. It is next to the Gower Peninsula which is well-known for its impressive landscape. Another, more important reason to come to Swansea is the impact of the department. Swansea claims to be amongst UK's top ten Computer Science Departments as it got a '5' in the last RAE and an 'excellent' in the Teaching Quality Assessment.

1.2 'Introduction to Computing' for Non-computer Scientists

The course 'Introduction to Computing' aims at teaching students who are not majoring in Computer Science. They used to be taught together with the Computer Science majors, however two years ago the department felt the need of separating the students into two groups. Furthermore, the course is open to students from any department interested in Computer Science as a 10 or 20 credits elective course.

The course has two parts, Introduction to Computing I, taught in the fall term, and Introduction to Computing II, taught in the spring term (compare also the module proforma A.2 and section 3). Each term has 20 lectures, i.e., two lectures each week, and 15-20 hours of labs in small groups. This academic year, the numbers of students on these two courses are relatively small. I have about 40 students on Introduction to Computing I where approximately half of them have chosen it as an elective. Some students are very interested in Computer Science and have already a good basic knowledge, others have no computing background at all. According to a questionnaire handed out to the students at the beginning, 1/3 of the students on the course has no programming experience and very little experience with computers. On the other hand, some students have A-Level Computing background and decided to do a Joint honours in order to be able to study two subjects. These students should get a good foundation in Computer Science in order to be able to continue with their studies in Level 2. There are also some exchange students on the course who again are quite able students. This presents the lecturer with the particularly challenging task to teach at different levels so that each student can benefit.

1.3 Objectives of the Course

According to the module proforma (A.2), Introduction to Computing I 'gives an overview of the main topics and questions in Computer Science and enables students who are not majoring in Computer Science to reach a level of skill in programming such that they will be able to apply their computing knowledge in their other studies.' Introduction to

Computing II enhances the programming skills and also provides ‘a basic understanding of algorithms and data-structures’. So, the aim is to teach non-computer-science students a good overview of computing and programming. As the course is taught separately from the Computer Science majors, I have the opportunity to start at a lower level and work through the important topics at their level of understanding.

1.4 My Background

I studied Mathematics and Physics in Munich, but instead of becoming a secondary school teacher—as originally intended— I started a PhD in Mathematical Logic. Mathematical Logic is one of the links between Mathematics and Computer Science, and the work I did for my thesis could be also located in Theoretical Computer Science. My thesis - expressed briefly - aimed at exploring techniques for extracting programs from proofs. It makes extensive use of an interactive theorem prover, whose development and maintainance involves a lot of programming and computational background. My PhD was funded by the programme ‘Logic in Computer Science’ in Munich and was finished in 2003 while I was working for the EPSRC project ‘Domain-theoretic methods for program synthesis’ in Swansea. In 2004, I got a tutorship in the Department of Computer Science at Swansea University. Since then I have been teaching courses on Algorithms, Computability Theory, Cryptography and IT-Security, and Specification to Computer Science students, as well as Introduction to Computing to Non-Computer-Scientists.

1.5 My Teaching Philosophy

My background is naturally one of the influences in my teaching. I do research in Theoretical Computer Science; I am a Mathematician who changed to Computer Science; I come from a foreign country; and I am female. This shall be all reflected in the following which expresses some points of my teaching philosophy.

Get students interested in the subject One of my main aim is getting students interested in Theoretical Computer Science, in Computer Science, and in Science, in general. Of course, not all of this applies to each situation, but, coming from Theory myself it is my aim to make students interested not only in the practical applications but also in the foundations of Computer Science.

How do Computer Scientists think? Students from other departments on my course probably won’t proceed to work as computer scientists. However, I stress that, in later work-life, it may be very useful, to have some insight in Computer Science, having had the opportunity to get to know some computer scientists and understand their focus and way of thinking.

Make students succeed You only learn programming by doing it - step-by-step. Therefore part of my course are compulsory lab classes where students solve weekly programming exercises. The student can have as much support by my assistants or myself as they wish and finally can get all exercises ticked off. Thus, students are trained to try on their own, but also to ask for help, and make the experience to progress. The ideal, of course, would be that they face so much fun that everything else comes on its own.

Support female students Probably not much need to be said here. In Computer Science, there are far less female students and female students tend to have higher dropout rates than male students. Solving this problem is however a sophisticated task. Nevertheless, my presence in the department hopefully serves as a role model for existing and potential future students.

Try to do it right I have spent quite a while examining the aims of my course. I really want every student to take something away from the course. One of the essential questions for me was whether to teach programming in a procedural or object oriented style. Currently I teach both styles, telling students that they should be aware of the fact that there are different styles, however for some students this is quite challenging.

2 Content

This section is about the content of my courses Introduction to Computing I,II. As the portfolio is written in the context of how to teach introductory programming, I focus on the programming part and intentionally neglect some of the other topics.

Essentially, I start with an 'Objects first' approach, using the Book 'Objects First with Java BlueJ', by Barnes, Kölling [BK04], and later concentrate on the more computational aspects of programming.

2.1 Teaching Language

The programming language used in the course is Java. This was decided by the department and is in accordance to the fact that also the computer science majors start with Java. One third of my students had prior programming experience; however only one of them used Java before. For those students the choice of Java seemed to be very good. For the students with no programming experience and computer science knowledge, in particular, for those who only want to a 10-credit elective, Java as a first language is rather tough.

2.2 Topics studied

Below I give an overview of the main topics studied in the two parts (11 weeks each) of the course. The syllabus is written down in the module pro-forma (A.2). A more detailed overview of the topics for the first part of the course can be found by looking at the table of contents (A.3) of my summary notes. The notes were handed out towards the end of first term and summarize the concepts studied, list the relevant examples and give further hints for revision. They are supposed to go together with the slides given out in the lectures.

Introduction to Computing I

1. Introduction using BlueJ (Week 1-2)
2. Writing Classes (Week 3)
3. Solving Computational Problems (Week 4-5)
4. Object interaction (Week 5-6)
5. Collections (Week 6-7)
6. Inheritance (Week 8)
7. Programming without BlueJ (Week 9)
8. Overview on Input/Output (Week 10)
9. Revision (Week 11)

Introduction to Computing II

1. Recall method calls, arrays, loops (Week 1-2)
2. Parameter passing (Week 2)
3. Recursion (Week 3-4)
4. Algorithms: Searching (Week 4-5)
5. Algorithms: Sorting (Week 6-7)
6. Data structures (Week 7-8)
7. Error handling (Week 9)
8. Revision (Week 10-11)

2.3 Literature

The book used for Introduction to Computing I is

1. David J Barnes, Michael Kölling, Objects First with Java, A Practical Introduction using BlueJ, Second edition, Pearson Education, 2004.

In Introduction to Computing II, I used

2. Cai Horstmann, Big Java, 2nd Edition, Wiley, 2006.
3. Michael Goodrich, Roberto Tamassia, Data Structures and Algorithms in Java, 4th edition, Wiley, 2006.
4. Robert Sedgewick and Kevin Wayne, Introduction to Computer Science, http://www.cs.princeton.edu/introcs/home/.

I recommend 'Big Java' as background reading because it contains the material for both parts of the course. For the next time, I plan to start with the first two chapters of 'Objects First with Java', however then immediately introduce the 'main' method and go on with material from 'Big Java'. This has the advantage that students are used to writing 'complete' programs, i.e. programs which run in any environment. Another reason for this decision is that some of my students are later doing a modelling course, which is built on 'Objects First with Java'. This way, I avoid too much overlap with that course.

2.4 Programming IDE's

At the beginning of the course I am using Java BlueJ as an IDE as I am starting with [BK04] as course book. Students typically like to work with BlueJ. According to a small questionnaire handed out at the end of the course, all but one students think that BlueJ should be definitely kept.

Later in the course, after having introduced the main method, and, in particular, when talking about algorithms, I use DrJava. DrJava is a easy-to-use IDE which can be installed very quick. For the lectures, DrJava has the advantage that code and compiler output or user interaction can be seen at the same time. Thus, it is not necessary to swap interfaces while explaining a program or its algorithmic behavior.

2.5 Object Oriented vs Procedural

The first time I taught this course, I started in a procedural manner, following a course by Sedgewick and Wayne, developed at Princeton University. Only later in the course, I used 'Objects First with Java' [BK04] to introduce object oriented programming. My experience was that the students found the Princeton approach very mathematical, whereas they appreciated the BlueJ approach very much. This made me start with BlueJ first in this year. I still think that students should learn both object oriented programming and procedural programming, however I also admit that it is quite a lot to learn and digest. The reason to stress procedural programming as well is that algorithms are typically taught in a procedural manner, and that many formal methods which are discussed later in the scheme focus on procedural programs, and not on object oriented programs.

3 Instructional Design and Delivery

3.1 Contact Hours

The number of contact hours for my course has been decided by the department (see also A.2). Introduction to Computing I and II, are both taught over 11 weeks with 4 and 3 contact hours per week respectively. These contact hours subdivide into lectures (2 hours) and lab classes (2 or 1 hour).

3.2 Space

I noticed that the size and layout of the lecture halls significantly influences my teaching in lectures. For instance, in Introduction to Computing II I had one a flat floor lecture hall, which simply looked like an old fashioned class room and a more common tiered, fixed seat lecture hall. It was quite obvious to use the former for tutorial like meetings, whereas in the latter I have presented new material in form of the usual lectures. One of the lecture halls used in Introduction to Computing I can be seen in appendix A.1.

3.3 Computer Labs

The size of the laboratories available for my course, made it clear the practical work only can be done in small groups. As my contact hours for this course are limited, this includes that the labs mainly need to be supervised by postgraduate assistants.

3.4 Delivery

As mentioned earlier, the main aims in teaching introductory programming is to show the students how to learn to program. Therefore, decisions about lectures and labs are mainly guided by this aim.

In the lectures new concepts are introduced, often motived and explained by means of examples. The use of Java BlueJ supports this approach and makes new concepts and examples very accessible to students. This involves programming in class from scratch, but (more often) modification of existing code in class.

3.5 Lab Classes

In each lab session students get various small programming problems to solve. The objective is to revise the examples studied in the lectures, and to start writing own programs. After finishing a task students ask the assistant to get the exercise 'ticked off'. (The lab exercises count 40% towards the continuous assessment.) Knowledge and own programming skills are built-up, right from the beginning. Students are forced to look at the material studied in the course each week. They also learn to work in class, to interact and to ask for advice.

3.6 Slides, Handouts, Board, Blackboard

In my course I have been using slides taken from the Java BlueJ book, combined with my own slides. The purpose of using slides in my course is twofold:

1. To keep an overview of what has been covered in the lectures.
2. To explain theoretical concepts.

The slides are handed out to the students at the beginning of each lecture, so that they have the opportunity to add comments and collect the material for later reference. Sometimes I leave space on the slides/handouts which the students will fill in during the lectures (see also appendix A.4).

Besides slides I often use the board, e.g.,

1. To fix or stress important points for later reference.
2. To explain examples in detail or to solve a problem which I have given to the students during the lecture.
3. To give an overview of the topics in the lecture.

Finally all the handouts in the course are available on Blackboard. There, students can also submit the coursework and find marks assigned to them.

4 Assessment

The assessment of the courses Introduction to Computing I and II, consists of two parts: continuous assessment (worth 50%) during term time and a written examination (worth 50 %) at the end of the term. The continuous assessment is further subdivided into lab exercises, assignments, and (in Introduction to Computing II) open book tests.

4.1 Continuous Assessment

The continuous assessment is supposed to support the step-by-step learning process. I personally prefer many small tasks instead of one big application, mainly to give the students the possibility to watch their progress and have immediate feedback. Actually, it is also valuable feedback for myself. I learn what students found easy, what difficult, and I can react upon this.

4.1.1 Lab Exercises

The students get weekly programming exercises, directly linked to the lectures. Ideally the students solve the exercises in the lab class following the lecture. They demonstrate their programs to an assistant in the lab who then ticks the exercises off. There is no further grading on the lab exercises. Overall this process ensures that students have done the work on their own. The exercises count up to 40 % towards the continuous assessment. The advantages are

- Knowledge is built-up step by step.
- Students receive immediate help when they have difficulties with getting started.
- Students receive immediate feedback.
- Ownership of coursework is clear.
- In order to get exercises ticked off students need to connect with staff. Therefore they are used to talk to staff which will simplify the process of getting help when needed.

Appendix A5 contains are examples for lab exercises set in Introduction to Computing I. Whilst Exercise1 directly refers to examples from [BK04], Exercise 4 was set as lab exercise after having introduced computational problems. A typical exercise sheet in Introduction to Computing II contains the task of implementing an algorithm given in pseudocode. Later in the course, students are also expected to design algorithms on their own, but that would be a typical task in an assignment, as opposed to a lab exercise.

4.1.2 Assignments

Assignments are to be solved by the student in their own time and are electronically submitted. Typically there are two assignments in each part of the course. The assignments give the students the opportunity to apply what they have learnt in the small exercises to larger tasks.

Appended (Appendix A5) is assignment 1 of Introduction to Computing II which is about implementing algorithms with different run-times. These will be used later as examples when discussing runtime of algorithms. The second assignment in Introduction to Computing II consists of a bigger application: implementing a cryptographic tool that encrypts and decrypts using various algorithms, ranging from simple ones to more complicated ones.

4.1.3 Open-book Tests

In Introduction to Computing II, I have included two open-book tests, one in the middle of the term and one at the end. Although counting very little (approx 10 of the the continuous assessment), the open-book test gives valuable feedback to the students and me. The students find out where they are and I get to know which concepts need to be explained more thoroughly. Finally, the open book test serves as a good preparation for the written examination at the end of the term.

In future, I also would like to include a practical test where students can demonstrate their abilities to interpret compiler outputs, to read the Java documentation, etc. By now I had not done so due to practical reasons such as that it is difficult to get all students together in labs at the same time.

4.2 End of Term Examination

4.2.1 Written examination

The written exam consists of both theory and practice. I mainly ask the students to design simple programs and test their knowledge on concepts. Having concentrated very much on programming during the term, the final exam makes the students also revise the other topics covered in the course.

The exam typically has three questions, the students are supposed to answer two out of them. Attached, as an example, is Question 1 of the last exam (see A.5). For setting examinations it is good to select an area which is quite natural for students. Then, every student can completely concentrate on the programming task and its solution, and is not distracted interpreting the task. Therefore, one of my favourites is to set a computational problem involving a clock.

A final word on writing programs on a paper: I generally think that students are spending too much time in front of the computer instead of thinking of an algorithm first. So, it does no harm to ask them to write down a program on a examination paper. Of course, since there is no copy and paste facility, nor a way to approach libraries, I will mainly test basic programming skills such as implementing a simple algorithm, writing loops, understanding a recursive program.

4.2.2 Supplementary

My supplementary examination for Introduction to Computing I, II is a practical examination. The main reason for this is that students won't leave their preparation for the supplementary examination until the last moment when they know that they have to be familiar with programming in front of a computer.

5 Evaluation

5.1 End of Year Module Evaluation

At the end of each term students are asked to fill in a Module Evaluation Questionnaire (see Artefact A.6). The lecturer evaluates the questionnaires as well as the assessment results and reports to the Learning-and-Teaching Committee of our Department where each course will be discussed. Below I analyse the results for the course Introduction to Computing I which was taught from September 2005 to January 2006. The questionnaire was handed out in the last lecture.

5.1.1 Questionnaire Results

In the questionnaire the students gave marks out of 5 where 0 is a negative response and 5 is a positive response. The marks below are the average marks (rounded).

Aims			
Understanding of Aims	[3.8]	Achievement of Aims	[3.7]
Notes			
Readability of Notes	[3.8]	Usefulness of Notes	[3.6]
Lectures			
Interest of Lectures	[3.5]	Usefulness of Lectures	[3.8]
Coursework			
Clarity of requirements	[3.8]	Usefulness to understanding	[4.1]

Student's comments on 'Positive aspects of the module' included 'lots of practical classes', 'well explained assignments', 'plenty of support', 'use of BlueJ and DrJava'. Some individual comments: ‘Comfortable class setting made it easy to ask questions or ask for help’, ‘Lab classes because you learn more by doing the work after the lectures’, ‘Lab classes: Practical experience with a safety net’.

Student’s Suggestions for Improvement included ‘No 9am lectures’, ‘have lectures in lab’, ‘more examples in handouts’, ‘reduce speed’, ‘less Maths’, and ‘Clearer introduction to how to code with java out of BlueJ before moving away from BlueJ’. Of course some of these suggestions for Improvement are beyond my control, e.g. the lecture times or the size of the labs which makes lecturing in labs in general impossible. Other points, such as the level of Math involved are definitely things which I will take into account when revising some work for the course.

5.1.2 Course results

The results of the course are subdivided in the continuous assessment during the course and the written examination at the end of the course.

	Failed	3rd	2nd	1st
Continuous Assessment	12 %	8 %	33 %	47 %
Written Examination	15 %	29 %	35 %	21%

Overall I am quite satisfied with the results. The number of students who failed seems to be a bit high, however this figure includes a few students who stopped attending the course but didn’t suspend. The numbers of firsts on the continuous assessment is also quite high, simply due to the level of advice available by my assistants. Some students simply would try so long until they get the questions right. Thus, in my opinion, the abilities of my students are best reflected in the examination marks. I had quite a few very able students who definitely deserved their good marks, and, as I could see in the examiners’ meetings, had comparable marks in their other courses.

5.2 Conclusion

It was a good experience to participate in the Disciplinary Commons. It showed me how much teaching is influenced by the context of a course, but also how much we are typically concerned with ‘what we teach’ and not ‘why we teach it’ or ‘how we teach it’. I became aware of the fact that not all of the problems I faced were due to my lack of experience; but simply were inherent to the task of teaching introductory programming. This concerns the choice of the programming paradigm, but also problems such as ‘how to respond to the individual needs of students’. My participation in the Disciplinary Commons made me change quite a few things in the course. An example of something which was really easy to change was to ask for more small, quick feedback. With respect to the future - in particular since I am teaching a course for Non-Computer-Scientists - I realized that

I can allow myself to teach less material, thereby focusing on the skills I really want to get over to my students. Finally, having seen how fruitful it is for a course to reflect on the decisions and to discuss them with others, I hope to keep this habit, not only for the teaching on the Introductory Programming course.

References

[BK04] David J Barnes, Michael Kölling, Objects First with Java, A Practical Introduction using BlueJ, Second edition, Pearson Education, 2004.

[Bro05] Glenn Brookshear, Computer Science, An Overview, 8th ed, Addison-Wesley, 2005.

[DD03] Harvey M Deitel, Paul J Deitel, Java How to Program, Fifth edition, Prentice Hall, 2003.

[GT06] Michael Goodrich, Roberto Tamassia, Data Structures and Algorithms in Java, 4th edition, Wiley, 2006.

[Hor06] Cai Horstmann, Big Java, 2nd Edition, Wiley, 2006.

[SW06] Robert Sedgewick and Kevin Wayne, Introduction to Computer Science, http://www.cs.princeton.edu/introcs/home/

A Appendix

A.1 Artefact: Sense of Place

A1-SenseOfPlace.pdf gives impressions of the campus at Swansea University. The picture at the bottom left shows the lecture hall where I taught Introduction to Computing I. The picture to the right has been taken in the computer lab of our department. The remaining pictures show our university, located close to the sea.

A.2 Artefact: Module Proforma

A2-ModuleProforma.pdf contains the official descriptions for Introduction to Computing I and Introduction to Computing II, as they have been approved by the university and can be found in the departmental handbook.

A.3 Artefact: Contents of Introduction to Computing I

A3-Content-IntroductionToComputingI.pdf contains the contents of the summary notes written for Introduction to Computing I.

A.4 Artefact: Sample Lecture

A4-Lecture-IntroductionToComputingII comprises the slides for a lecture, given in Introduction to Computing II.

A.5 Artefact: Assessment

A5-Exercises-IntroductionToComputingI.pdf contains two sample exercise sheets handed out in Introduction to Computing I. The first exercise sheet builds on Java BlueJ, the second focuses more on the computational aspects in the course.

A5-Assignment-IntroductionToComputingII.pdf is the first Assignment of Introduction to Computing II. Whereas the exercises just need to be done and shown in the labs, assignments need to be handed in and get marked.

A5-ExaminationQuestion-IntroductionToComputingI.pdf comprises the first out of three examination questions for Introduction to Computing I. Students were meant to attempt two questions out of three.

A.6 Artefact: Module Evaluation Questionnaire

A copy of the Module Evaluation Questionnaire is handed out to students at the end of each course.