Manifesto for COSC326

Discard your expectations. This paper is quite different from any other course you have taken so far. There will be no lectures, and there will be no obvious teaching, but you will undoubtedly learn a great deal and you will have to work very hard. We will introduce some elements of 'real life', where you will be the computer professional with the answers, who guarantees the client a working product, and where the client must be able to trust your competence.

Programming includes two important elements—analytical skills, and creative problem solving skills. For this reason, a great deal of time in this course will be spent on developing these skills through a series of problem solving assignments. You will be given problems and you will be expected to continue with a problem until you have solved it or satisfactorily completed the exercise. Some of the assignments will not require use of the computer, although you may find the computer useful for ancillary tasks.

Requirements for Attendance and Assessment

COSC326 is a purely practical course. There are two two-hour supervised laboratories every week and a single one-hour 'Town Hall' meeting where we will discuss problems and analyse approaches. We will also use these sessions to drop hints, give out instructions, and listen to your ideas. *You are expected to attend all of these sessions*; they hinge on your participation. You should also be prepared to spend five to nine hours each week outside the scheduled labs in reading, preparation, and unsupervised laboratory work. If you do not attend and put in the work, you are not likely to pass.

Assignments

Programming is a practical skill and you can learn only by practice. We cannot give you 'real' programming problems because these take weeks or months for professionals to complete. Instead the course consists of a series of smaller problems to be solved in the laboratories. These exercises have been carefully chosen to make you discover and develop certain skills. In music teaching, such studies or *études* have been used for exactly this purpose for hundreds of years.

In real programming projects, part of the problem is cleaning up the defects in the specification you get from your client. You need to be alert to this because we do not guarantee that our problem specifications are clear or exact. We encourage you to ask for clarification of a problem whenever you are unsure about instructions and *before* you embark on programming.

You will do some assignments as individuals or in pairs and others in groups. Generally a group can solve problems that are beyond the abilities of the individual members.

You will also find the workload may be decreased through effective collaboration. To work most effectively in a group, you need to learn how to cooperate. Working in pairs or groups is an integral part of the course and you will be required to participate in these. However, even when assignments are completed by a group, we will reserve the right to monitor individual contributions, participation, and ability level in order to satisfy the pass/fail criteria of the course. We also reserve the right to ask for specific individual contributions.

Pass/Fail assessment

The course is 100% internally assessed, and is a Pass/Fail paper. There are a number of reasons for this. First, we wish to foster learning, not test what you already know; we are more interested in the process of solving problems than in the final outcome. Secondly, some skills are impossible to grade. And thirdly, the notion of a 50% pass mark and an 80% 'A' grade can sometimes lead to dangerous situations in the world of practical applications. You would not be happy, for example, if you thought an aeroplane pilot could land safely only 80% of the time.

In this course a program that works 'more or less' is not good enough. Computer Science professionals are expected to be able to solve problems, and real software is expected to meet basic professional standards. For this reason, the passing criterion for COSC326 is that you complete each assignment to a 100% satisfactory standard. The assignments, however, are designed also to facilitate learning and stretch your ability. For this reason, we may sometimes accept as satisfactory a solution that is less than perfect. Sometimes it may become clear that one or more of the assignments was 'too hard' or, in a particular case, that no more useful learning can be obtained from it even if it is not complete. For this reason, we may have to vary the details of the assessment scheme as the course progresses.

Assignment Scheme

There are 14 assignments to which we have allocated 25 points. You are expected to score 25/25 to pass. Therefore, these points are really only indicative of the rough size of each assignment.

Submission of Assignments

We do not expect you to hand in the assignments at the end of each lab, but you are advised to try and keep up with the work. You will find that in most cases you will be asked to do more work to complete an assignment and you need to leave time for this. No submissions of assignments will be accepted after *the last day of lectures*. Most

assignments will be assessed in the laboratory sessions. Some will be submitted electronically; you will be given instructions on how to do this during the labs.

Course Objectives

This course aims to meet an extensive set of learning objectives with the overall goal of helping you to become more effective programmers through problem solving. After each étude you will see a list of numbers. These refer to skills in the list below, and specify the principal skills that the étude is trying to foster. These are not distributed evenly. Some skills are needed for almost every task and some simply require more attention because they are harder to develop. The most important generic skills are having a good understanding of a problem and working with people and these are therefore implicitly part of the objectives of almost every étude.

1. Understanding a problem

1.1 simplifying

1.2 clarifying

1.3 generalising

1.4 specifying

2. Problem solving strategies

2.1 lateral and creative approaches to problem solving

2.2 solving problems according to specification

2.3 top-down and bottom-up solution designs

2.4 solving a related problem

2.5 working backwards

2.6 choosing appropriate tools, e.g. pen and paper, spreadsheet, programming language

2.7 applying persistence and thoroughness, removing mental blocks

2.8 appropriate, effective, and creative use of resources (time, personal abilities, library etc.)

2.9 simulation, finding metaphors, trial and error

2.10 designing and using appropriate notation

3. Computer related techniques

3.1 understanding the limitations and problems specific to computer program execution (such as overflow, rounding errors, division by zero)

3.2 understanding file formats, platform specifics and proprietary aspects of computing

3.3 understanding recursion and iteration and when and why to use them

3.4 using appropriate data structures, considering suitability, simplicity and size

3.5 adequate testing, debugging and validation

3.6 understanding efficiency, profiling and measuring

4. Working with people

4.1 understanding different approaches required in team and individual projects

4.2 becoming aware of group dynamics; identifying different skills of group members, task allocation, participation, conflict resolution

4.3 collaboration and ego-less programming

4.4 working with other people's programs; reading, understanding, debugging, maintaining, re-using code, libraries

4.5 producing software that is adequately documented, commented, and user-friendly

4.6 an understanding of ethics, and professional responsibilities towards colleagues and clients, including confidentiality and software protection

4.7 recording proceedings, decisions, progress etc.

4.8 writing clear and concise reports