

Department of Computer Science

COMP1100-99-Review

Unit Review

1 Preamble

This is an introspective document written by the lecturers for the unit (BPM, HG, MCN). Additional review material will be provided by course surveys.

2 Comments

The following points are made in no particular order. The first person refers to BPM or HG.

Orientation Laboratories These were a major distraction at the start of the year. The material for the so-called “Olabs” was in quite awful shape. The material was a mish-mash of system information (of some value as reference material) and laboratory exercises (which are particular to the start of the year). Some effort was spent by BPM and Wiefa Liang in trying to partition, rationalise and bring the material up to date. In retrospect this was time that was stolen from mainstream COMP1100 development. The “Olabs” in week 01 were cancelled because of computer system difficulties, and they were concentrated into week 02.

It was a major distraction to explain to students that enrolment in “Olabs” was different to enrolment in COMP1100 tutorials and laboratories.

The “Olab” arrangement meant that the first effective laboratory session in COMP1100 did not take place until week 04. Week 05 was compromised by Easter. The semester was essentially half-way through before any sort of systematic laboratory and assignment momentum was in place.

The recommendation for 2000 is clear. While an orientation laboratory scheme makes sense, it should occur purely inside COMP1100. The previous arrangement, of being extra-unit, made sense in earlier years when we had in first semester four whole-year units running in parallel. We now have only two first semester units (COMP1100 and COMP1200) and to a sufficient approximation the student group of the latter is a subset of the former.

The introductory laboratories should address the various tools that they need to do the course: emacs, emacs in Eiffel-mode, netscape, news, email and introductory unix commands. They could include an introduction to shell command line commands.

Textbook The textbook (Weiner: An Object-Oriented Introduction to Computer Science using Eiffel) turned out to be of limited effectiveness. It was a lukewarm decision at the time, with optimism just outweighing the misgivings. His overall structure and placement of emphasis was appropriate for the unit and some of his ideas and explanations are quite good. The devil, however, is in his details which are occasionally quite sloppy. His code segments cannot be trusted and needed to be frequently corrected in lectures. His introduction to Eiffel syntax was too brief. In the event the optimism was misplaced. The book could not be used as a source of detailed exercises (see later).

An alternative, rejected narrowly, was the material put together by Robert Rist of UTS. This had a strong information systems flavour which struck no chords whatsoever with the lecturing group. It also had a range of errors and suffered from sloppiness in the text.

What to do next year? There appears to be no suitable textbook for the unit, among the books on the market (other texts are too advanced). One option is that our revised lecture notes should be the text for next year (with Weiner and the other references being optional reading).

How did it go? The final reported results for the unit are given in the next table.

	COMP1100	ENGN1213	total	percent
HD	44	24	68	18
D	63	16	79	21
CR	68	16	84	23
P	44	08	52	14
PS	10	01	11	03
N	37	01	38	10
NCN	21	00	21	06
WD	16	02	18	05
	303	68	371	

In the end it went surprisingly well. The failure rate of those sitting the final examination was 10%, and 11% either withdrew or didn't sit the final examination.

We seemed to have only a small amount of wastage, that is, students who withdrew before the HECS cutoff date. (This is a figure that does not seem to be reported to lecturers.)

What did we get done? The original unit objectives included the following.

- In the context of an Eiffel program involving several classes and library classes
 - systematically determine the run-time behaviour of the program,
 - write a narrative description of selected program properties.
- Given an appropriate specification, modify the behaviour of a given program within its design and architecture

- Carry out program exploration and modification
- Communicate the concepts of an object-oriented programming language
- Define the main aspects of the engineering of software systems.
- Define the basics of data structures and algorithms.

The match of course outcomes with these objectives seemed to be pretty good. The objectives were achieved (in full or in part) for the large bulk of the students. The course was perceived as being interesting by many students - as evidenced by the large and well-behaved attendance at lectures through to the end of the semester. The major criticism has to be that the translation of these objectives to actual programming skills was poor.

Lecture Material Some major misjudgements were made with how students learn computing in general and object-oriented computing in particular. BPM gave a first section, which was taught in a way that BPM would have liked to learn computing. Roughly, by establishing a robust framework before hanging detail on it. Wrong! Most students did not engage this. They were taken outside their comfort zone, and they gave quite clear feedback that it wasn't working. BPM responded by changing tactics, and HG and MCN put together their lectures in this context.

All lecture slides were prepared for overhead projectors, using \LaTeX and published in Postscript and PDF. All lectures were developed *ab initio* and not surprisingly were not available in advance of the lecture delivery.

An initial objective to present lectures from a laptop via the video projectors was abandoned. In the event the brightness of the video projectors in the MCC lecture theatres was judged to be unacceptable.

What might be done about the problem next year? The situation can be rectified with a simple restructuring and augmenting of the extant lecture notes. I think that it is great to start the course by asking the question "What is a computer?" but that the models given should not include objects. The discussion of Eiffel syntax, routines, structured design and algorithms (which would occupy the large bulk of the course) should be formulated with respect to single class systems. Finally, objects and OO methods could be introduced (along with their machine model) in the last third of the semester. This restructuring could include all of the lecture material of COMP1100.

Disparate Backgrounds This is the standard and perhaps unsolvable problem with a first unit in a discipline, which does not have an entry prerequisite.

- A group of (self-assessed) skilled students emerged towards mid semester, requesting extension work. The problem was solved by putting them in touch with Andrew Tridgell. Part of the group dropped out (not being as skilled or as enthusiastic as they first thought they were). The rest have been enthusiastically constructing game engines.

- A (larger) group of computing novices were handled in several ways. BPM gave walk-up consultations that helped a group of students who were otherwise in danger of getting left behind. The real find was Fergus McGlynn who gave a series of “catch up” tutorials. Fergus was recognised by all students in COMP1100 and he served as the human face of the unit.

What might be done about the problem next year? There seems not to be any silver bullet.

- It is surprisingly difficult to reform the small-group sessions on a skills basis, once the semester is under way. Tutors have timetable constraints, students have timetable constraints, rooms are in short supply. If streaming is to be done, it would seem that it has to be done right at the start of the unit.
- I resist the notion of setting up an “honours” unit. As far as I am aware only Mathematics still does this, and that is based on the precise metrics from secondary school, namely the Advanced Maths Extended Double Major stuff. An honours unit in first semester only begets the demand for a subsequent honours unit in second semester.

The Eiffel Experience The experience of working with `Eiffel` was positive. The simple PDE provided by `small-eiffel` (GNU) and `emacs` proved effective, and the commercial ISE environment was not used. We chose not to pay the overhead of moving to a new environment part way through the semester. Students found the syntax quite intuitive, and the gap between concepts and their realisation in `Eiffel` was quite small.

My assessment is that the decision to use `Eiffel` has been supported by the experience of COMP1100, and that DCS should confirm its commitment to use it as the backbone of its teaching enterprise.

Assignments We had the problem of starting with essentially a zero base of assignment archives. In approximate terms, the assignment format involved students in completing elements of two applications (a coin-weighing game and a bridge-hand generator). Suffice it to say that the assignments proved to be effective (almost all students “got them done” and none seemed to give up on the unit because the assignments were at the wrong level).

At the start it was anticipated that students would work within a gui-based context. This proved to be difficult within the `Eiffel` environment used, for technical reasons. In the end the assignments were still characterised by character-based input-output. We were not proud of this.

The assignments were successful in the sense that most students completed them but it is questionable whether they were successful in teaching programming skills. In my opinion, it was wrong to hit them with a large multi-class system for the first assignment. This can wait for the last part of the course. I am not particularly fussed that the GUI did not come about and do not think that the students were either. There is plenty of interest and plenty to do apart from working with GUIs.

Tutorials and Laboratories The unit decided to run a one-hour tutorial and a two-hour laboratory session each week (for 09 weeks of the 13). I judge this to have been effective but very inefficient.

- The demand on the academic staff to produce suitable material was horrendous.
- Tutorials remained marginally effective. It was probably better to have them rather than not to have them, but they represent a very poor return on investment. Put bluntly, students do not much know how to utilise tutorials and we do not much know how to run them. For much of the time both they and us would rather be somewhere else. It remains unproven as to why a teaching technique developed and perfected in the humanities is directly applicable to the newest of the technologies.
- The laboratory sessions took a while to find their feet at the Eiffel level. The laboratory environment remains a chaotic one despite our current attempts to control it. Students come and go at will, equipment problems (while small) are enough to prevent the enforcement of an invariant of the form “you shall do this in the session and you shall hand it in”. The key dilemma is that most laboratory sessions provide a task environment that enables student learning but does not enforce student learning. At the farcical end students say that they “did” the tasks but made no attempt to engage the material. They type in the input, but do not read the output. These tasks are effective if the demonstrator works with the student: “explain to me what is happening here” but ineffective otherwise.

What didn't we do? It became painfully clear towards the end of the unit and during the examination marking that some students did not have basic skills. For example, when given the text of a simple routine (which they had worked with in an assignment) and asked multiple choice questions at the level of “what is the initial value of an integer local variable” they were very unsure indeed. These students had successfully completed assignments. (This does not mean that the assignment was not their own work).

It seems to me a simple fact of life that for a nontrivial number of our students the ability to answer such questions will only be developed by forcing them to do a large number of “micro” exercises. The fact that many exercises will only differ trivially from one another is a positive feature, as the students are unable to extrapolate from one situation to an analogous one. They appear to be ineffective in using a unifying general principle to derive the answer in a specific situation (such as applying the parameter passing mechanism).

This is neither good nor bad: it just is. It means that if we want to test them on exit by asking such questions then we had better adopt a teaching environment that develops the ability. That means lots of exercises, hundreds of them. Otherwise we are simply condemning some hard-working and earnest students to failure.

The role of machine support (including marking) needs to be considered, with tutor intervention for students who need it. In this scenario, tutorials are a targeted activity for a class of students, not a universal activity.

The lack of programming skills and knowledge - even for quite simple single class programs - was quite distressing and must be improved next year. I believe that this can be done by restructuring the course as above and confining the first, and possibly part of the second assignment to being single-class exercises. These first assignments should have several short parts and should test the knowledge that they are meant to have obtained in labs as well as lectures. The final assignment should be a multi-class system. There is a place for the drilling of concepts in labs and tutorials.