SOME GOOD IDEAS FROM THE DISCIPLINARY COMMONS

Sally Fincher
Computing Laboratory
University of Kent
S.A.Fincher@kent.ac.uk

David Barnes
Computing Laboratory
University of Kent
D.J.Barnes@kent.ac.uk

Peter Bibby
Computing & Electronic Technology
University of Bolton
P.Bibby@bolton.ac.uk

Jim Bown
Complex Systems
University of Abertay, Dundee
J.Bown@abertay.ac.uk

Vicky Bush
Multi Media & Computing
University of Gloucestershire
vbush@glos.ac.uk

Phil Campbell
Computing
London South Bank University
campbep@lsbu.ac.uk

Quintin Cutts
Department of Computing Science
University of Glasgow
quintin@dcs.gla.ac.uk

Stephan Jamieson
Department of Computer Science
Durham University
stephan.jamieson@durham.ac.uk

Tony Jenkins
School of Computing
University of Leeds
tony@comp.leeds.ac.uk

Michael Jones
Computing
Bournemouth University
mwjones@bournemouth.ac.uk

Dimitar Kazakov
Department of Computer Science
University of York
kazakov@cs.york.ac.uk

Thomas Lancaster
Department of Computing
UCE Birmingham
Thomas.Lancaster@uce.ac.uk

Mark Ratcliffe
Computer Science Department
University of Wales, Aberystwyth
mbr@aber.ac.uk

Monika Seisenberger
Department of Computer Science
University of Wales, Swansea
M.Seisenberger@swansea.ac.uk

Dermot Shinners-Kennedy
Department of Computer Science
University of Limerick, Ireland
dermot.shinners-kennedy@ul.ie

Carole Wagstaff
School of Computing
University of Teesside
C.A.Wagstaff@tees.ac.uk

Linda White
School of Computing & Technology
University of Sunderland
white.holmes@sunderland.ac.uk

Chris Whyley
Department of Computer Science
University of Wales, Swansea
C.J.Whyley@swansea.ac.uk

Disciplinary Commons Web Page: http://www.cs.kent.ac.uk/~saf/dc

ABSTRACT

In this paper, we describe the Disciplinary Commons project and identify some practical ideas which address central issues for teaching and learning of introductory programming that have emerged from it.

Keywords

Introductory Programming, Scholarship of Teaching and Learning (SoTL)

1. INTRODUCTION

The *Disciplinary Commons* is a project whereby teachers come together to share and document their practice through the production of course portfolios. In the academic year 2005/6, 18 teachers of introductory programming courses in different institutions met together every four weeks to discuss and document their teaching. This kind of forum is unusual in Higher Education, and a number of collateral benefits were discovered as

we worked towards the goals of the project. Firstly, the group encountered examples of problems in multiple courses being addressed in similar ways. Secondly, the group discovered examples of unique practices which addressed common problems. This paper examines both types of example.

2. PROGRAMMING ENVIRONMENTS

Initial teaching of programming (itp) is often accomplished within a particular environment. Often these are commercial tools, appropriated for educational purposes. Here, we present three different approaches to supporting students with environments designed for novices.

The first, Vortex at Wales Aberystwyth, focuses on the design of object-oriented programs. The second, BlueJ [3] at Kent, provides a completely integrated development environment to support visually the development of programs. The third, SNOOPIE [4] at Abertay Dundee, offers support with both program and problem formulation and can link in to any configurable IDE.

2.1 Vortex

The VorteX [1, 2] environment is a UML based, text editing environment aimed specifically at novice software developers. Developed in house, the software places much more emphasis on object-oriented software design than is typical for students at this stage of development. The resulting environment is a full Java 1.5 complaint toolset that supports both single and group projects. A useful tool for students, but the real power is what it provides the academics. VorteX records everything that students do during their development (adding, removing, editing and even chatting). It is then able to generate statistics on who did what, when, how much individuals contributed to a project and what difficulties they had. It is an ideal environment for maximising student feedback without generating extra work for the academic.

2.2 BlueJ

BlueJ is an interactive Java environment that integrates a text editor, the standard Java compiler, an interactive debugger and also has support for JUnit-style unit testing. BlueJ is used for two main reasons. First, program creation via a text editor and then compilation and execution via a separate command-line interface can be difficult for novices. In particular, Java's classpath idiosyncrasies make for confusion and inconsistency between different command-line environments. Most of the students are not familiar with command-line programs, whereas they are typically familiar with GUI-based programs. Second, the tutor concerned wants to teach Java in an objects-early (preferably objects-first) fashion. An IDE offers the best chance to provide students with object visualisation, which is believed to be essential to them getting the idea of what objects are. While supporting the full Java language, the interface it presents to users is deliberately uncluttered and easy to use. Furthermore, its visualisation of objects, visual distinction between classes and objects, and the ability to inspect object state and call methods interactively are very powerful supports for an objects-first approach.

2.3 SNOOPIE

SNOOPIE recognises two fundamental problems that novices have in developing programs: first formulating a (working) program at all and second formulating the right program to address the problem. Compiler error messages are notoriously obscure, and to assist program formulation SNOOPIE captures those errors and expands them with text related to the current teaching material, drawn directly from dialogue with students. Messages thus encapsulate both the compiler error and an extension sensitive to the novices (note, extensions provided may be changed over a term). SNOOPIE also parses the program for common (semantic) errors, for example ';' at the end of for and if statements and failing to update loop counters. SNOOPIE also provides more sophisticated support in the way of problem formulation. It is able to parse a student program and identify the presence or absence of key components at any degree of granularity, for example 'a void method called x that takes 2 int parameters' and 'a nested for loop where the inner loop repeats 3 times and the outer twice'. Moreover, these program checks may be structured to allow progressive support through an exercise.

3. SMALL-BUT-OFTEN ASSESSMENT

Using any environment, practice of programming skills is vital. A lightweight assessment model of small, weekly exercises with frequent but limited (yes/no) feedback is promoted at Swansea and Abertay, Dundee. These exercises, designed to be adjacent to the lecture material and so familiar and likely to lead to success, have two side-effects. First, students are in an environment of continual activity and the exercises are of sufficient simplicity and consequence to promote collaboration among those students who find programming challenging. Second, the need to engage with the tutor on completion of one or more exercises on a weekly basis ensures regular dialogue between staff and student.

This dialogue additionally helps ensure ownership of work, and understanding of what has been done. Where ownership or understanding is in question, students may be asked to extend the submitted work to provide some additional, simple, functionality. (see section 6, below)

4. STRUCTURED REFLECTION

Every introductory programming teacher recognises the value of students' reflection on the process of their learning, using meta-cognitive and self-explanation strategies. Within the *Commons* several ways to encourage these behaviours have been encountered.

4.1 Logbooks (i)

At Kent logbooks are used as a self-motivated learning aid for students. They are expected to obtain a physical A4 logbook at the start of the course and makes notes in it on every occasion when they are doing practical programming work. They are advised to note objectives for the session, and then to make reflective notes on the actual outcomes of that session. Class material typically makes suggestions for things they might like to record in it.

They are explicitly told that the logbook will not be assessed in order to free them from concerns about keeping it neat and presentable, and to enable them to organise it in the way that best suits their particular learning style. However, in order to reinforce that we take keeping a log seriously, class supervisors are asked to initial the log at every practical session and answer any queries students might have noted since the last session. As the logbooks are checkpointed in this way, the students can be invited to submit a logbook—as supportive evidence in defending a plagiarism allegation, for instance.

Reactions from the students are mixed, but generally positive. Some make too close an association between practical work and assessment and cannot see how a permanent record might be of use once an assessment is passed. Others enjoy the freedom of being able to record what **they** feel to be the important topics covered in the course.

4.2 Logbooks (ii)

At LSBU students are expected to make an entry into their logbook every time they do some work on a terminal which contributes towards the assessments. The advice given to students requires their entries to:

1. Indicate what they intend to achieve.
2. Indicate what they did to achieve it.
3. Indicate what problems they had achieving it and how they overcame them.
4. Indicate why they decided to attempt a particular path through the learning material.
5. Indicate what questions they have on what they have done.
6. Indicate what they intend to do next.
7. Every so often reflect on what they have achieved.

Students are also advised not to waste time on:

- Making it neat (as long as it is readable)
- Copying the course material (unless it is related to the problems they are having).

An important concept is that the logbook should not be a notebook. That is it should not contain summaries from the text book or from the web material. It should not contain program designs, listings, or scripts of program runs. It is an executive summary of the students' learning activities. It is not their notes from the lectures. (These may all go into the back of their book.)

Students should make an entry in the logbook every time they do a significant activity connected with the unit, not just during the weekly practical classes. They should bring their logbook to every practical class so that you tutor can assess and sign it. Students are advised that logbooks will not be assessed or signed at any other time.

4.3 Self-assessment grid

Bolton has a series of Good Things which are asked for in their assessments—meaningful identifiers, sensible use of functions etc. And they hope students will supply them. And if they don't, their tutors will give them feedback.

Self-assessment grids make it easier to identify Bad Things and target the appropriate feedback. Over time, they may help students spot the Bad Things for themselves.

The grid simply consists of three columns.

- The first column is the "wish list";
- The second column is for the student to indicate whether or not they've achieved this (they can simply tick or put in a comment);
- The third column is for the marker to supply their comments.

Good Thing	Student	Lecturer
Meaningful identifiers	Yes	"Wombat" for loop control?

White Space	Yes	Where? I couldn't find one blank line!

Whilst the grid itself isn't marked, it is mandatory—no grid, no assessment mark. With the physical grid, it's easy to spot students' misconceptions, as well as their sins of omission and commission.

5. How to Start a Lecture

5.1 Go note-free

It is rare to go into a lecture these days without seeing a Powerpoint presentation. Careful use of such tools can lead to excellent presentations but these are rare in the academic arena. Many tutors do not have time for such elaborate preparation. The end result is often an endless set of slides that look only marginally better than traditional overhead transparencies. Such lectures are often boring to listen to; and are almost as boring to give.

The approach used at Aberystwyth is simple; don't use slides except when displaying complicated figures, tables or photographs. This is not as radical as it may first seem. The suggestion is to talk with, not at, the students. Tutors can provide lecture notes, book references, *etc*. Despite popular belief students will read this material so long as they are motivated. Nothing destroys motivation more than having to sit through endless hours listening to an academic reading their slides.

5.2 Harness technology

Before a typical lecture starts, students tend to come into the theatre, sit down, and then talk among themselves about a wide range of subjects - but rarely the one about to be covered in the ensuing lecture. Capturing the moment, at Glasgow the lecturer displays a question ready for the students to answer as they settle themselves. This question will usually address some aspect of the last lecture. Each student is asked to record their opinion as to the answer using an electronic voting system. There are a number of benefits:

- the students (who all have a voting handset for the year) get it out of the bag, ready for this and future questions
- their attention is drawn towards the front of the lecture theatre, to the question
- their thoughts and conversation, for a while at least, will be on the subject matter of the question, bringing to mind content from the last lecture
- their interest is raised right at the start of the lecture, since they typically enjoy seeing if they got the question right, and how other students answered

The choice of question is important. It can be used to open a review of a topic the students found hard in the last session. Or as an opportunity to open up a new topic. Like any question, it should be unambiguous, particularly because the lecturer has less chance to resolve misunderstandings in the hubbub at the start of a lecture. In theory, the technique could be used successfully without handsets, although significantly more students attempt to derive an answer for themselves when the results of their efforts are displayed in aggregate and discussed by the lecturer. Students like immediate feedback!

5.3 An analogy, a paradox, a puzzle

At Limerick, the lecturer usually tries to start each lecture with one of the following: (1) an analogy (2) a paradox, or (3) a puzzle. The idea is to try and get the class 'thinking' about the topic that is about to be discussed. It is probably more correct to say that the tutor wishes to get them reasoning about the topic so that they start to form a view (their view) about it.

Analogies are a common and well documented source of knowledge transfer. The closeness of fit between the target concept and the analogous concept is crucial and it is often difficult to get a very close fit. Consequently it is important for the tutor to manage the consideration of the analogy and attempt to keep the focus on the aspects that match the target concept. A useful one is:

When introducing basic data structures like stack and queue the tutor asks the class what happens to new text messages they receive on their mobile phones. Students have been observed to immediately point out how the newest message goes in at the start/beginning/front and becomes the first accessible message in their message list with the other, existing messages being 'pushed down'. They even offer justifications for this and in a sense feel obliged to 'defend' the strategy. The tutor subsequently ask them what happens to voice mail messages. The students advise that new voice mails are added to the end of the list of existing messages and again they are quite happy to provide an explanation as to why it is different from the text messages scenario. These interactions provide a useful platform for putting names on these mechanisms (i.e. a stack and a queue) but the principal advantage is that the students believe they have provided the rationale for these mechanisms and that in some sense they have ownership of them because the mobile phone technology provides everyday experiences of them. From a pedagogic perspective there is no need to make a case for having a list that behaves like a stack or a queue, and no need to justify simple operations like insert at front and remove from front only.

6. INDIVIDUALISED ASSESSMENT

For reasons of academic integrity and to ensure that students are suitably prepared for further study it is necessary to ensure that the programming assessments they submit are the results of their own effort. There are several heavyweight processes to identify plagiarism [5]—some involve running all solutions through a detection-engine, others asking every student to describe how the code they have submitted works. An interesting alternative is to produce programming assessments that are unique for every student.

6.1 Personal Input

At Gloucestershire, assessed exercises are varied each semester to reduce the possibility of plagiarism by taking work from a student who took the module in a previous run. Where possible, exercises have an element of individuality. For example, one introductory programming module asks students to implement an animated screensaver. To personalise this, they have been asked to choose a representation that relates to a hobby or interest:

*Your **requirements** are to design and create a pattern or picture that shows the illusion of some simple animation or movement, such as you might see in a screen saver. The image should have some personal significance e.g. it could be your initials or a logo related to your favourite music or football team, for example.*

In other years, they have been asked to choose scenes related to a particular topic e.g. Spring, cartoon characters and Outer Space. Because the students must design the scene themselves, it is difficult for them to copy from each other.

6.2 Free choice

A key principle at Durham is to enable independent, self directed learning. Tutors support personal interests, objectives, prior experiences and learning preferences. One way of enabling this, is what is called the "December Project".

Students are told on arrival in October that they have the opportunity to conduct a personally chosen (formative) project. They are invited to submit details of their choice so that resources can be provided. Choices are also made public for the sake of those starved of ideas.

The project runs for the last fortnight of term one. The students are encouraged to continue to work on it over Christmas and, if they wish, during the first week of term two. Completed projects are assessed by staff in one-to-one discussions during laboratory classes.

The project is a successful way of "individualising" learning. It's clear from the one-to-one assessment that many students do take ownership and consequently give an honest account of their level of attainment. They are proud of their achievements, and where let down by lack of experience, keen to discuss how to progress.

6.3 Re-combination

Individualisation does not need to solely occur within the programming parts of an assessment. It can also include other elements of the software development process, such as design, testing or critical reflection.

As an example, the software development team at UCE Birmingham use an Open Office document template, merged with the contents of a simple spreadsheet and individualised with an OO-Basic script. The open source software is used as PDF files can be produced, eliminating many disguise strategies used by students on contract cheating sites. The most recent assignment involved producing a playable simulation of the TV show ‘Deal or No Deal’. Individualisations issued included the strategy used by The Banker, the sections of the solution for which a pseudocode design or class diagram was required, or the areas for which a detailed test log was needed. Care was taken to ensure that the same learning outcomes were tested for all students and that all deliverables were at the same level of computational difficulty.

In this way tutors ensure that colluding students cannot directly copy from one another; the work they submit has to have some original components to it. Further the unique combinations mean that work placed on a contract cheating site [6], a site where students place work out to tender, can be traced to an identifiable student.

7. SUMMARY

By presenting these examples, we hope that they might be of interest to colleagues in similar situations. In this way, we work to extend the community of the *Commons*, sharing and documenting our practice to make it available for future use and development.

8. REFERENCES

[1] Thomasson, B.J., Ratcliffe, M.B., and Thomas, L.A., Identifying Novice Difficulties in Object Oriented Design, Eleventh Annual Conference on Innovation and Technology in Computer Science Education, University of Bologna, Italy, June 2006

[2] Ellis, W. and Ratcliffe, M., An Anonymous Approach to Group Based Assessment, 8th International Computer Assisted Assessment Conference, Loughborough, July 2004

[3] www.bluej.org

[4] Coull N, Duncan I, Archibald J, & Lund G. Helping Novices Interpret Compiler Error Messages. 4th Annual LTSN-ICS Conference, 26-28, 2003

[5] Lancaster T. & Culwin F. (2004), A Comparison of Source Code Plagiarism Detection. *Journal of Computer Science Education* 14.2, pp101 - 117.

[6] Clarke R. & Lancaster T. (2006), Eliminating the successor to plagiarism? Identifying the usage of contract cheating sites. To appear in *Proceedings of JISC International Plagiarism Conference 2006*, The Sage, Gateshead, Newcastle, UK, 19 - 21 June 20