

Two Pedagogical Patterns for Course Design

Joseph Bergin

<http://csis.pace.edu/~bergin>

berginf@pace.edu

February, 2003

This paper contains two recently discovered pedagogical patterns. The first, Student Tasks First, is a candidate for inclusion in the Patterns for Active Learning and the second, Differential Grading, is a candidate for inclusion in Feedback Patterns. Both involve course design. Student Tasks First tells how to begin the course design process and Differential Grading gives advice about how to design the grading/assessment structure. This latter advice is consistent with the Bloom Taxonomy.

The EuroPloP 2003 shepherd for this paper was Christa Schwanninger, who kindly and carefully helped me refine the ideas as well as the presentation. Thanks for a wonderful exchange and a lot of expert advice.

Version 4

Student Tasks First

Joseph Bergin

You are designing a new course or revising a course. You are at the beginning of your deliberations on the shape of the course. You have an incomplete idea about what the students should learn in the course, but you believe in **Active Students** [2].

How do you begin the design of a new course?

You have a difficult task with many things to do. You have to decide on what you will teach, how you will teach it, what the students will do, and how you will assess them. Coming up with a complete set of learning objectives first can be a formidable task with many tradeoffs. You might try this and find later that you have been too ambitious. On the other hand, you know you want your students to be active throughout the course and to engage in activities that will teach them the key ideas.

Therefore, first decide on what the students will do in the course. Design the student tasks. You can then ask what they will have learned if they complete these successfully and make adjustments as necessary. If it is insufficient or the wrong thing, you can add or drop exercises from your list. If it is too ambitious, you can make adjustments. The process can be iterative, with each cycle driven by exercises you design or find.

You should consider a range of activities from short in-class exercises to large projects if appropriate, including closed and open labs (See note). It is not enough to focus on what they will read or hear. What activities will necessitate their reading a particular thing? What is it that they can do to solidify and validate their knowledge? "Patterns for Active Learning" [2] has more to say about the kinds of activities that you should consider.

After you have a set of student tasks, your course design can then proceed to determining what you must do to facilitate success in the student tasks. This can help you design lectures and other course support materials. If you accept that most learning will occur through these activities, then you also have a basis for designing assessment of the students. You can not only simply assess the tasks, but you now understand what they should have learned by doing the activities, so you can assess that.

If you interpret it broadly, as "**do** the important thing first," then this is similar to Early Bird [1] applied to the course design process. The most important thing in a course is what the students will do in the course, not what the instructor will do. So do that first.

When you design the lectures first you are focusing on yourself and not the students. It is very easy in this mode to plan to do too much. What you do has less importance than what the students do in any case, as the name of the game is learning, not teaching. You want your students to be active [2] so design this part first.

If the course is new to you then you can mine the web for exercises of other educators or search the candidate textbooks for project suggestions. Other educators at your own

institution or that you can communicate via email are also a good source of ideas for exercises, as are the proceedings of conferences of educators in the field: SIGCSE in the case of Computer Science.

This advice was suggested by the author during a panel in which he participated at SIGCSE 2003: [3].

Note: In U.S. usage, a closed lab is an exercise that is done with a group of other students under the close direction of an instructor. They occur at a fixed time and in a fixed place. Open labs, on the other hand, are done by the students according to their own schedule and in a place of their choosing.

[1] Bergin, J.; Early Bird, in Fourteen Pedagogical Patterns; Web: <http://csis.pace.edu/~bergin/PedPat1.3.html>; and in the Proceedings of EuroPLoP 2000

[2] Eckstein, J., Bergin, J., and Sharp, H.; Patterns for Active Learning; Web: <http://csis.pace.edu/~bergin/patterns/ActiveLearningV24.html> and Proceedings of PLoP 2002.

[3] Huggins, J., Bergin, J., Caristi, J., Walker, E.: Survivor: Getting Through that Class the First Time; Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education; February 2003, p 234-5.

Differential Grading

Joseph Bergin

You are designing exercises and the grading structure for your course. You want to make the course relevant for all your students.

You must design the course so that all the students will grow, not just the average student.

You recognize that your students have a wide range of ability. If you target your exercises at the average student they will be impossible, or at least very frustrating, for some and boring for others. You want to teach them all. In fact you want each student to reach his or her own potential. However, you don't want to categorize the students or make assumptions about them ahead of time.

If you ask students to do things above their ability they may not learn much and will be frustrated. If you ask students to do things below their ability they may not learn much and will be bored. You want to stretch all your students, of course, but not until they break.

Therefore, provide graded assignments: **C level** assignments, **B level** assignments, and **A level** assignments. Use the Bloom taxonomy of assessment [1] to design these exercises. **C level** exercises stress *knowledge* and *comprehension* of the material. **B level** exercises stress *application* and *analysis*. **A level** exercises stress *synthesis* and *evaluation*. The student must complete all exercises satisfactorily for the desired grade, but need not do work at a higher than the desired grade. Exercises at the **B** and **C levels** are pass/fail and may be repeated. **A level** exercises may be quite open ended under an individual contract with the instructor. Evaluation exercises at the **A level** can involve evaluation of other student's work.

The grading suggestions correspond to adjacent pairs of levels of the six level Bloom taxonomy of assessment. **B** and **C level** exercises can be tested objectively using well-designed measures [4]. **A level** exercises need to allow for creativity and individuality. The instructor should require that **C level** exercises be completed successfully in addition to **B level** exercises for a **B** grade. Similarly, for a grade of **A**. If programming exercises are used at the lower levels, then quality requirements must be absolute. "The program must run correctly, ... "

Within the exercises for each level there are at least two different activities. This is primarily because we have combined two Bloom levels to achieve each grade level. It also has the benefit of permitting adaptations to different learning styles, as there is more room for variety.

Bloom states that before you can perform at one level you must have achieved all prior levels. Thus for the **C exercises**, you have a knowledge component (a multiple choice test, perhaps) that must be successfully completed before the second comprehension component.

Note that the individual student selects the level at which to perform. A (hypothetically) weaker student can first establish a base line and then work to extend his or her knowledge and hence the grade. Also, the weaker passing students have at least achieved something

successfully rather than being assigned a grade because they performed poorly on everything. Once achieving a base level attempts at a higher level will not affect the grade negatively. This can be a boon to a student trying hard to extend him or herself.

The lower level exercises may be repeated by the student, perhaps multiple times. Between attempts it is important that the students get appropriate feedback [2] that will enable them to move forward. Un-graded exercises should also be provided to the students to aid their learning before they attempt the assessed activities. Feedback other than grades should be provided for these.

With a good collection of objective exercises for the lower grades and interesting and challenging projects for the **A level** objectives, you can emphasize student team work, peer feedback, and other group exercises without undue effort on your own part and with increased learning on the part of the students. The Feedback Patterns [2] have more to say about these topics.

Good objective tests are difficult, but not impossible, to create. For this pattern to work, only non-trivial questions should be considered. They must be carefully crafted to adhere to the Bloom levels. See Lister [4] for more advice on how this can be achieved in Computer Science and related fields.

A consequence of using this scheme is that you set different goals for different (self-selected) ability groups.

This pattern is drawn from work of Lister and Leaney [3] who call this method "criterion-referenced" grading.

[1] Bloom, B.S., et al. Taxonomy of Educational Objectives: Handbook I: Cognitive Domain, Longmans, Green, and Company; 1956.

[2] Eckstein, J., Bergin, J., and Sharp, H.; Patterns for Feedback; Web: <http://csis.pace.edu/~bergin/patterns/FeedbackPatterns.html> and in the Proceedings of EuroPLoP 2002, Irsee, Germany

[3] Lister, R and Leaney, J; Introductory Programming, Criterion-Referencing, and Bloom; Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education; February 2003, p 143ff.

[4] Lister, R.; Objectives and Objective Assessment in CS 1; Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education; February 2001; p 292ff.