Assessment Tools based on Bloom's Taxonomy of Educational Objectives

Nanette Veilleux
Boston University

Abstract

Fair and useful assessment of student abilities is often a difficult task. Ideally, evaluation instruments should assess how well the student has understood material directly presented (knowledge and comprehension), how well the student can apply this information to a new problem (application), how well a student can distinguish and relate the component parts of a topic or argument (analysis) and how well a student can extend his/her learning to new areas (synthesis and design). These four skills represent different, progressive, levels of understanding, that fall along an abridged hierarchy as that outlined in Bloom's Taxonomy of Educational Objectives[1].

This paper describes a method of designing in-class exams and take-home projects for a freshman computer science course. Here, the design of the test questions and project requirements makes explicit use of this abridged version of Bloom's Taxonomy of Educational Objectives. The in-class tests described in this work evaluate the depth of a student's understanding by incorporating a planned variety of questions, ranging from those easily answered by a student who has understood basic lectures and reading to problems requiring novel application of basic tools. Based on the degree of difficulty of the questions answered, students are graded according to a deterministic criterion (as opposed to, e.g., scaling based on class averages). The take-home projects also employ a deterministic criterion that indicates precisely what is expected of the student for each of three performance levels: passing (C), good (B) and excellent (A). An important feature of this design is that the instructions for the lower level work are more detailed and require less student innovation, whereas the instructions for A level projects offer less direction.

I. Introduction

Developing useful and fair evaluation instruments such as tests, projects and papers are often cited in informal discussions among faculty as the most difficult task in teaching. On the pre-college level, tests are traditionally graded according to a strict numerical mapping from percentage correct to a letter grade. In college, it is more typical to convert the percentage correct to a letter grade, based on the distance from a mean, loosely (and usually incorrectly) referred to grading on a bell-shaped curve. While this serves to provide a distribution of grades by normalizing performance to the context of the immediate class rather than assigning an a priori mapping (e.g. 87 is a B), this method is empirically as well as statistically suspect. In practice, "grading on a curve" is often based on a small sample size and the variance of the sample is disregarded. Even if the statistical analysis is correct, normalizing for all bias factors

present in a particular section of a particular course in a particular year narrowly sets the standard as the present class's average ability. The instructor-related context is conflated in this grading scheme. Instructors who have over- or underestimate a class's ability, or who have simply provided poor instruction, can smooth over their misjudgment by using a curve.

An additional difficulty with grading on a curve becomes apparent to instructors of introductory classes. For example, students arrive in an introductory computer science class with a wide range of previous experience. While it is preferable to shunt sophisticated programmers to a more advanced course, many incoming freshmen's backgrounds have sufficient theoretical gaps to keep them in an introductory course. However, judging novices in direct competition to their more experienced cohorts does not reveal either group's progress or the merit of the course.

A more meaningful standard and one that can overcome these biases can be derived by using evaluation methods that focus not on relative performance among students, but on the absolute performance. It is important, though, to design test instruments carefully in order to evaluate relevant student performance. Grading based on performance also sets a more meaningful context for each student. Instead of competing among themselves, they are given the objective goal of performing according to a broader standard: e.g. any students who successfully complete a freshman calculus course should know how to integrate and differentiate reasonably complex equations. These standards can be clearly articulated as course goals and used to motivate students. Young students, in particular, are often confused about the expectations and usefulness of a course which is addressed by clearly stated objectives.

II. The Use of Bloom's Taxonomy in Test/ Project Design

On the most general level, performance-based tests and projects are to be designed to evaluate "what a student knows" about the subject material. On closer examination, "knowing the material" is a complex accomplishment, with many dimensions. Students might be able to recite every fact they're read or heard in the course and so demonstrate a breadth of knowledge on the topic. On the other hand, a student may be able to use a few well-understood principles and from these develop novel solutions to topical problems.

Expectations of students' performance will vary with the level of the course. The goal of a training course (e.g., how to use MatLab) might be to impart a large number of facts to the trainee. An introductory course might expect students to comprehend basic principles, to apply these principles to similar problems and to identify the components and interrelationships involved in these principles. An upper level course, relying on the foundations set in lower level courses, would require students to use these principles to solve complex and unfamiliar problems and to be able to evaluate other solutions critically.

Ideally, evaluation instruments for an introductory course such as tests, projects and papers, should be able to assess how well each student has understood material directly presented (knowledge and comprehension), how well the student can apply this information to a new problem (application), how well a student can articulate underlying assumptions, principles and relationships (analysis). More ambitious criteria could also assess how well a student can extend

his/her learning to new areas (synthesis). These four skills represent different, progressive, levels of understanding, that fall along a hierarchy abridged from the one outlined in Bloom's Taxonomy of Educational Objectives. Bloom et. al. describe the major taxonomical categories as: knowledge, comprehension, application, analysis, synthesis and evaluation.

In this paradigm, students who have mastered the first stage of learning (knowledge) are able to recall material presented directly in class lectures or readings. A student who has understood the material can use it to solve new but familiarly stated problems (comprehension). A student has achieved a deeper understanding of the material when he or she is able to independently solve a problem posed in an unfamiliar way (application). A student would next integrate class material well enough to be able to independently see the relationships between the parts of a topic and use this understanding to analysis the underlying process or hidden assumptions in a problem (analysis). Ultimately, a student should be able to use this deeper understanding to produce a novel solution (synthesis) and critically evaluate solutions (evaluation). Since these skill levels are achieved progressively, tests can be designed to evaluate and assign a grade based on the highest level a student has attained.

Mastering the higher levels in Bloom's hierarchy (synthesis and evaluation) is a more appropriate expectation in upper level rather than in introductory courses. While the principles of designing tests in the framework of Bloom's hierarchy can be applied to any course level, the examples given here are from introductory courses. They illustrate how to design instruments to evaluate comprehension, application and analysis skills. Consideration is also given to testing synthesis skills in order to challenge the student's emerging understanding.

A. In-class exams

There are several well-known difficulties in preparing in-class exams. The most notorious is probably the limited time frame. Usually tests must be administered in a class period, which may be as short as 50 minutes. Many universities schedule finals for longer limited intervals, such as two or three hours, with the assumption that the entire course will be covered in the final. Instructors often focus on coverage and consider an exam comprehensive if every major topic covered in the course is represented in the test. If coverage is seen as an issue that focuses on the breadth of knowledge, this paper offers an alternative method which considers and evaluates the student's depth of knowledge.

Depth of knowledge can be framed in terms of Blooms taxonomy. Examples from an introductory computer science class and from a calculus class are used to illustrate the implementation of test design based on the abridged hierarchy (comprehension, application, analysis) described above. Synthesis level testing is desirable even in an introductory class to give students experience if not proficiency. Evaluating whether a student can perform on the synthesis level is more easily tested in take-home projects and will be described in the next section.

<u>Comprehension</u>

Although these three levels of understanding (comprehension, application, analysis) appear straightforward, the assessment value of a specific test question is, of course, situationally

constructed and depends not only on the subject material but also on details of the specific instruction and student background.

For example, in an introductory computer science course, a student might be asked to trace a program similar to ones seen in class, and calculate the values of variables along the flow of program execution.

```
int  x = 5;
if (x ==5)
   y = x * 2 - 3;
else
    y = x - 4;
```

This is an example of a question that tests 'knowledge/ comprehension'. Here, the student recalls what has been taught about arithmetic statements in a computer language, precedence, selection structures, etc. and uses these facts. The student demonstrates knowledge of terminology, methodology, etc. and also comprehension of program execution as it proceeds along one path (the x==5 is true path) rather than another.

Another computer science example that students often find difficult, but still tests only their comprehension of the course material, is to trace the value of x after the statement:

```
int x = 5;
x = x %2;      // % is the modular or remainder operator in the C language
               // x %2 would divide x by 2 and return the integer remainder
```

This would require a student to have understood something more basic about the nature of integers, the properties of the binary modular operator and the recursive use of the variable x. (Although it is unlikely that the student would be able to state the nature of the problem in these terms.) The difficulty in this problem is the unfamiliar operator % and a weak understanding of arithmetic that is demonstrated when students are unable to extend principles of binary operators to unfamiliar operators.

Another example, drawn from an introductory calculus class, would be to ask the student to perform basic derivatives such as d[2 cos x] /dx. Presumably, a similar example has been covered in class as well as in the text and homework. The student is only required to remember this fact and reproduce it faithfully.

Application

A student achieves a deeper understanding when s/he is able to assemble the facts learned in class and apply them to a new, unfamiliar problem. These questions would probably be similar in difficulty to routine homework exercises, and require a deeper level of assimilation of the material that simply recalling what has already been demonstrated. In an introductory computer science course, a student might be able to answer more difficult questions such as:

MicroHard Computers is producing new chips that address memory using 5 times

powers of two. Fortunately there's a function in math.h that will calculate $5*2^n$ for any nth power of two: ldexp(5,i). The following program has been written to print out addresses from 5 to 5120 ($5*2^{10}$). Unfortunately, it has at least five bugs in it. List the bug and its fix in the spaces below:

```
#include <iostream.h>
void main ( ){
        int i;
        int return;
        while (i<9)
                return = ldexp(5,i);
                cout << "5 times 2 ^ " i = return << endl;
} // end o' main
```

In this question, the student must apply what has been taught about data values, reserved words, library functions, output and loop syntax, and loop iterations without being directed to use or even consider these issues.

An example of an application-level question from calculus might be to find

d[cos(2x +3)] / dx .

Here a student must apply the chain rule, based similar examples from class instruction and text. Of course, if this problem has been covered well enough for the student to be able to produce it by rote, answering it correctly simply requires recall.

Analysis

At the far end of the taxonomy used in introductory courses here, an analysis-level question requires a student to fully understand the basic parts of a subject and how they interrelate. In a computer science course, the fundamental difference between integers and floating point variables requires a deeper understanding of memory allocation, type referencing and addressing. An example to test at the analysis level of understanding follows, where what a student knows about basic arithmetic is challenged by the program's output:

```
    ...
void main(){
    double x = 22.0/7, save_x;
    save_x = x;
    x = x +  100000000002;
    x = x -  100000000002;
    if (x == save_x)
            cout << "x +/- 100000000002 is the same as its initial value \n";
    else
            cout << "x +/- 100000000002  is NOT equal the initial value of  x \n";
} // end of main
```

Explain why the output of this program is

x +/- 100000000002 is NOT equal the initial value of x

In an introductory calculus class, exam questions can also be used to assess the degree to which a student has integrated the basic concepts into their understanding. For example, the student might be asked to construct an argument justifying whether

d [cos x] / dx =
a) sin x or
b) - sin x,

a commonly confused recall issue. This question requires that the student understand fundamentally that the derivative is the slope of the function, and that the cos x has a negative slope over the interval x = 0 to x = pi. The sin x function is positive from [0,pi], whereas the -sin x is negative on this interval and therefore the correct answer.

Practical Considerations

Of course, whether specific test results demonstrate a comprehension versus application versus analysis level of understanding is situationally constructed: it's just recall if the student has been shown even a sophisticated example thoroughly enough to be able to copy it from memory.

On the other hand, in the recall example for an introductory Computer Science course, students and instructors alike assume that the student already has a well integrated, solid understanding of basic arithmetic concepts, such as addition and Boolean logic. These questions test arithmetic understanding at a level that would correspond to Bloom's analysis or perhaps synthesis level. However, as far as new concepts are concerned, tracing a simple program requires nothing more than recalling how basic arithmetic is coded in a computer language.

There are many debatable middle ground questions. For example, if the modular operator has been covered sufficiently enough to have been memorized by the students, a slightly more difficult question may be required to test whether the students understand its application. One can also explore the gray areas between comprehension and application level understanding, which often involve how much a student is patterning from class examples versus how much the answer is generated from a deeper understanding. An instructor, however, usually has an intuition about the relative sophistication of the questions given text and class coverage of a topic.

B. Take home projects

Satisfactory higher level questions are more difficult to develop for a time limited in-class test. Although introductory course students may not be expected to produce professional level output, some synthesis level understanding can be expected. These questions might be best addressed as part of a take-home project. This is especially appropriate in a computer science course where the reasonable goal is that students be able to write computer programs to solve problems. A student's development is progressive and it is important that the take-home project requirements are designed to assess at what level the student understands the material.

One helpful and important tool when designing a project that tests across level in the taxonomy is the notion of 'scaffolding'. 'Scaffolding' is the process of providing the student with examples or hints. Scaffolding can help set the context to test all aspects of Bloom's taxonomy with unseen material. For example, an instructor can provide more "hints", or descriptions for how to proceed (e.g. by including derived equations, referring to an example in class, suggesting programming structures in a Computer Science course) for parts of the test that have been designated to test "comprehension" or "application". Although scaffolding is used even in in-class tests, this principle can best be seen in the context of take-home projects.

In one midterm take home exam given in the Introductory Computer Science course, the following requirements are given for a student who wishes to get a 'C':

For credit in the C/C+ range, the program
a) must do (at least) everything specified above in the program goal,
(which is to input the areas of five cubes and find their volumes)
b) must be able to read in five numbers from a user (the areas of the faces of the five cubes).
c) must be able to calculate the edge from the area,
d) must be able to calculate the volume from the edge,
e) must be able to add the all the volumes together (use a variable to store the running total-- this is what the sum from i=1 to n volume_i means) and then divide by 5 to find the average volume,
f) must use at least one repetition or selection structure,
g) must be well-commented, correctly indented and readable,
h) must compile and run,
i) must be accompanied by a report of input and output used to test how well the program works,
j) must be accompanied by a disk with the source (*.cpp) code on it.

A large degree of scaffolding is present in the requirement description alone. Items b-e describe, recipe-fashioned, how to find the average of the volumes of a cube, given the area of a face square. A student who has comprehended the material should be able to write the required program without much innovation. However, the B and A level instructions contain additional requirements but less or no scaffolding regarding how to meet these new problem specifications. For example,
For credit in the B-/B/B+ range, the program
a) must do everything the C-level program could do, as well as
b) must be reasonably robust to unseen data (should have a warning if an overflow or other fatal error might occur),
c) must be able to throw out "bad"" input, but be able to keep prompting until the user finally puts in a total of five ``good'' inputs,
d) must use repetition and selection structures when possible.

Here, the B level program states requirements beyond those outlined in the C level program but not the methods to satisfy the new stipulations. A student must generalize from the programming tools they've learned and apply them independently. Finally, the A level program requires more

difficult additional steps and provides less guidance for the solution.

For more credit (A-/A range), the program
a) must do everything the C and B-level programs could do,
b) except it must also find the average volume of any specified number of cubes
c) must be able to terminate the program if the user inputs more than three bad entries and output whatever average volume of the remaining "good" cubes (if any).
d) must be able to calculate the standard deviation .

For example, in the B program, information is given about possible unseen data that might be problematic (overflow or other fatal errors) but does not specify what "bad" data is. (E.g. a cube's area should never be negative and taking the square root of a negative number will cause a fatal error). The A program specification is still more cryptic in its additional requirements. In this example, the program must be flexible enough to count and divide by any number of good data points entered, either in total or before three bad entries. The instructions do not warn that there will be a fatal error if no good data is entered. In that case, most programs will have zero input data points and will divide by zero finding the average. The formula for the standard deviation is given in class or earlier in the program, but the necessity of storing or re-entering the data to calculate the standard deviation after the average is not specified.

III. Writing the test and assigning the grade

The first step in creating an evaluation instrument that follows Bloom's Taxonomy is to start with explicitly and clearly outlining minimum course objectives, both in terms of breadth and depth of understanding. For example, one might decide that no student should receive even a low passing grade if s/he cannot calculate elementary derivatives by the end of an introductory calculus class, or trace a simple program with functions and arrays after an introductory computer science class. The first important step, then, is to include questions on the exam that will test the minimum requirements. In technical fields such as computer science and engineering, mastering comprehension of simple principles is in itself a significant and appropriate minimum task. This can be measured by relatively easy comprehension-level questions that may be taken directly or nearly directly from in-class exercises, or homework solutions. This does not make the test "too easy" but allows the instructor to discriminate between the struggling students and the absent ones. Grades given for answering these questions would range from F (unable to answer more than a few questions) to perhaps a C-/C (able to answer all of the comprehension level questions).

The next step is to develop questions that provide information about a more satisfactory level of skill students might have attained beyond this minimum standard. The expectations of subsequent courses' prerequisites can give insight into what skills a student will need to be able to apply independently and what analytical skills they should have acquired when they arrive at this next stage. For example, if a student is able to independently design computer programs of sufficient complexity to fluently code the necessary components of a stack or queue, then the student is prepared to move into data structures. In this case, tests and projects should allow an instructor to determine, and a student to demonstrate, that the student can code and trace arrays,

functions and control statements when appropriate. This skill level corresponds to application level understanding, and a student who is able to demonstrate ability at this level would earn a C to a B grade.

In the same way, a subsequent calculus course or a Signals and Systems course can guide the goals of an introductory calculus class. If the student is able to expand basic functions into series, then the mechanics of Fourier and other transforms will not be the barrier to exploring these orthogonal transformations. A student who demonstrates proficiency on application level series, derivative and integral questions would earn a B grade.

Finally, using open-ended questions (that still have specific answers) is usually a good way to test whether a student has integrated the class material into a cogent analytical framework. Asking the student to provide "why" or "how" along with "what" is a straightforward way of doing this, as shown in the analysis examples above. These questions can have relatively short answers, which should have been anticipated and preassigned grading criteria. Students who are able to answer these questions correctly would earn a B+/ A grade.

One of the most difficult aspects of test design is finding a balance between different types of questions that require different levels of understanding. Although elementary school teachers who use outcome-based instruction can often give students a series of gateway tests, where a student must pass one level to be instructed and tested on subsequent levels, this is usually not practical in a university setting. Instead, instructors can give mixed exams using a variety of graded- difficulty questions. If an instructor explicitly writes enough comprehension questions (e.g., short questions with a 40% point allocation), enough application questions (e.g., less directed questions with a point allocation of 40- 50%) and a few analysis-level questions (remaining 10- 20% open-ended questions) to ensure that students who have mastered each level can demonstrate their performance, then grading becomes easier and less controversial. Assigning greater weights to more difficult questions amplifies the spread, allowing an instructor to more easily see, from points alone, where a student falls in the hierarchy.

Following this design, students with an analysis understanding will be expected to answer most of the comprehension and application level questions and some degree of the analysis level question, thereby earning 80 - 100% or an A. Students who have not developed a deeper understanding of the underlying components but are reasonably adept at applying the material in straightforward but unscripted problems will answer the comprehension and some application level questions, earning between 40 - 80% (e.g. C: 40-60/ B:60 - 80). Finally, students who are unable to answer any questions that require innovation will earn 0 - 40% (F / D).

Concerns arise about special cases where a student might correctly answer the most difficult questions and miss the ones designed to test recall. In eight semesters of tests for several courses, including introductory and intermediate programming, calculus and a graduate signals and systems course, such a student has not materialized, in a significant way. Students who have been able to answer the questions requiring analysis (e.g., the significant digit loss in the float example), have been able to answer all comprehension and application questions, with a few

exceptions due to carelessness. Including more of the less demanding questions mitigates the effects of infrequent carelessness through the point allocation.

It is important to carefully design instructions and questions so that they are not misunderstood, and early errors do not impact other questions. If there are sufficient numbers of all types of questions, occasional sloppy answers to easy questions or lucky answers to difficult questions will not affect the overall score significantly. Of course, the most difficult questions (in these examples, analysis-level exam questions or synthesis level in the projects) are best framed in a way so that "shot in the dark" answers won't suffice.

Here, giving open-ended questions with specific, anticipated answers and grading criteria usually addresses this difficulty. For example, in the x +/- 100000000002 problem above, no points are given if the student provides only the simple explanation that the output is produced if the if-statement is true (straightforward comprehension of if-statements answers this). Instead a student must explicitly mention that the double variable x has limited precision which is exceeded by x +100000000002 and not recoverable by subtracting 100000000002. Mention of fourteen significant digits, and its relevance should also be anticipated and given more credit.

By explicitly designing tests to include a preselected mixture of questions that test comprehension, application and analysis, students of all levels of understanding can demonstrate their abilities. Interestingly, in large introductory classes, taught by an experienced instructor, with a class composition that is roughly random, one might actually expect the performance distribution is approximately Gaussian. The difference between this approach, and an a posterior bell shape curve assignment is that grade levels have been determined based on a satisfactory level of performance (The criterion above would map satisfactory but not yet proficient application level performance to a C+/ B-). In introductory courses with no gateway criteria other than college admission I find that the distribution is broadly speaking bell-shaped with an average that tends to fall near this satisfactory mark. However, it is possible for all students to earn high or low grades.
One final tangible benefit from an objective based grading criterion is that there is less disagreement with students over their grades and there is less need for arbitrary post-facto grade adjustments on the instructor's part, a sign of a fair assessment tool. Because the grading criterion maps grades to meaningful levels of understanding, the grade distribution can be used to guide instructors in adjusting the level of instruction for new courses. For example, if most of the students are able to apply and analyze the material given, course expectations can be expanded in terms of the number and difficulty of topics presented in subsequent semesters.

IV. Conclusions

In order to evaluate a student's performance in a course, instruments such as tests and take-home projects must be designed according to clearly stated course objectives that include measuring both the breadth and depth of a student's understanding. Depth of understanding can be framed along the taxonomy outlined by Bloom and test question and problems can be written to assess a particular skill level on this taxonomy. The entire test or take-home project will include a variety of questions so that a student any where along the hierarchy can demonstrate their

proficiency at a particular level. Finally, grades can be assigned based on the performance on this variety of problems, which will correspond to success at a particular, progressive skill level.

Advantages of this methodology are an intrinsically meaningful grade assignment, less controversial grading criterion that re-iterate and reinforce overall course goals, and a test design process that encourages instructors to test what they ultimately wish to be teaching: knowledge, utility and insight.

Bibliography:

1. Bloom, Benjamin S., Engelhart, Max D., Furst, Edward J., Hill, Walker H. and Krathwohl, David R., *Taxonomy of Educational Objectives, Handbook 1 Cognitive Domain* David McKay Company, Inc. New York. (1956).

NANETTE VEILLEUX

Nanette Veilleux is an Assistant Professor in the Computer Science Department of the Metropolitan College of Boston University. She received her Sc.B. in Biophysics at Brown University and her MS EE and Ph.D. at Boston University in the Speech Processing and Interpretation Laboratory. Her current research focuses on statistical models of speech and language. In addition, she teaches traditional freshman in the Science and Engineering Program and non-traditional adult graduate students at the Metropolitan College. Dr. Veilleux chairs the Academic Policy Committee at the Metropolitan College, originators of a college-wide review of grading policies and practices. Course material is posted on Dr. Veilleux's Web site: http://metcs.bu.edu/~nmv