Tracking Changing Learning Goals (long version)

We (Anne Condon and Steve Wolfman) have been working on expressing learning goals for CPSC 101 and revising the course so students will more effectively achieve those learning goals. Our efforts in the course fall between the proposed level for core courses and for non-core courses. This is a brief tale of the JavaScript programming introduction in 101 and how it developed through two iterations of revision of the course.

In the process, we came to a much clearer understanding of what we needed students to learn from CPSC 101 (and crucially, what we did *not* need them to learn) and discovered some natural ideas about how to teach students to emphasize these goals.

Starting Point

A year and a half ago, students in 101 were expected to "learn JavaScript". Students got a more detailed sense of the learning goals from the textbook reading, from a small number of sample exam questions, and from our introductory JavaScript remarks:

- Learning JavaScript will introduce you to programming: the expression of algorithms in a computer executable language!
- JavaScript will illustrate to you central programming concepts such as: the importance of structuring data and key "control constructs".

We actually introduced the JavaScript language through two example interactive web pages. For example, here is the central code from the first example:

```
onMouseOver="document.images[0].src = 'logoUBC.jpg';"
```

After 2 hours of lecture, students worked through JavaScript challenge exercises for another 1.5 hours. We observed that students fared very poorly on these exercises, making little progress and often not even understanding the intent of the questions.

You can see our materials, including lecture notes, reading assignments, and sample exam questions, at http://www.ugrad.cs.ubc.ca/~cs101/2006s1/.

First Iteration

Last year, we reviewed the resources and assessments focused on JavaScript (among other things), trying to articulate our implicit learning goals for the unit. At the same time, we were also establishing a set of four high-level learning goals to guide the whole class, and we tried to connect up these detailed learning goals to the high-level ones. We added the learning goals we developed to our existing slides:

> After the "JavaScript" unit, you will:
>
> - appreciate the extra power to express processes of a programming language (vs. a markup language)

- understand a few key programming "control constructs", particularly events and conditionals
- know how to embed JavaScript in a web page
- know how and where to find JavaScript code
- be able to modify existing JavaScript code to suit your purposes

Note: the best way to learn to program is to program! So, do the lab and explore playfully beyond the bounds of the lab!

Our primary direct benefit from this exercise was clarifying our intentions for the unit, although phrasing the learning goals also suggested some small changes to the lectures. One indirect benefit was easier "hand-off" of the whole topic and specific subtopics to TAs and future course instructors through a clear statement of the main focus of the unit. Communication with the students also improved as we and they could "attribute" quiz and exam questions to explicitly stated goals, which made the exams easier to plan and allowed students to study more efficiently. (We see "studying to the exam" in the form of mastering the learning goals as a positive form of "efficiency", as opposed to overfitting to a particular practice exam.)

Unfortunately, these goals remain rather vague and difficult to connect directly with assessments. For example, what does it mean to "understand a few key programming 'control constructs'"? Should students be able to write code with these constructs? Read code with these constructs? Precisely model execution of code containing these constructs? All of these?

Second Iteration

We're currently in the process of a second iteration of the course learning goals, along with more substantial revision of teaching materials. For the JavaScript unit, we still found the connection between learning goals and exam questions was weak. Furthermore, our learning goals failed to emphasize what we considered the most important programming skill for 101 students: reading and tracing existing code.

So, we again rewrote the learning goals, this time with emphasis on the activities students engage in as they read and trace code. We split the new goals into three units that covered 4 hours of lecture time (about the same amount of time as previously used):

After Part I of the "JavaScript" unit, you will be able to:

- apply the "study/model/predict/experiment/refine" technique to learn new programming concepts
- connect the following terms to their use in programs: function, function declaration, function call, function name, function body, parameter, parameter list, variable, value, statement, return statement, assignment statement, expression, orthogonality, sequential execution (primarily textbook work)
- accurately model and predict the behaviour of variables in JavaScript programs
- accurately model and predict the flow of control in a JavaScript program through sequential execution
- employ the concept of "orthogonality" to combine your knowledge of different programming elements, including being able to interpret their combined effect

After Part II of the "JavaScript" unit, you will be able to:

- accurately model (and so predict) the evaluation of any expression, no matter how complex, as long as you have a good model of the parts
- model the construction and interpretation of expressions using numbers, strings (text snippets in quotes), arithmetic operators, and function calls
- use the <script> tag and events to add behaviours to your web pages (assuming you have a reference listing the available events and the behaviours you want are ones that you know how to implement in JavaScript)

After Part III of the "JavaScript" unit, you will be able to:

- accurately model (and so predict) the flow of control in a JavaScript program through a function call
- accurately model (and so predict) the flow of control in a JavaScript program through conditionals (both if and if/else statements)

These goals centred on the idea of building a mental model of how elements of a program work, applying that mental model to a new piece of code to predict its outcome, testing the outcome, and revising the mental model to accommodate the outcome. This led naturally to a classroom treatment in which we pose a series of small programming problems for student groups to solve, generally before any detailed discussion of the concepts used. We then discuss any different solutions, work with the students to articulate the models represented by each solution, experiment, and refine students' models. We're currently teaching the JavaScript unit with these goals and approach (3/4 of the way through).

Assessing these learning goals is fairly straightforward, since they were designed around processes that the students can master and display. For example, we expect to test the goal "accurately model and predict the behaviour of variables in JavaScript programs" with an exam question like:

Consider the following code:

```
var x, y, z;  // Line 1
x = num1;     // Line 2
y = num2;     // Line 3
x = y;        // Line 4
z = x + y;    // Line 5
z = z + 1;    // Line 6

alert("x is: " + x);
alert("y is: " + y);
alert("z is: " + z);
```

For lines 1-6 of the code, sketch the state of each of the variables after that line of code executes. Finally, what is the output of the program? [The real question would include delineated spaces for each sketch.]

This second round provided the same benefits as the first. In addition, rephrasing the learning goals suggested a concrete classroom approach that (based on formative in-class assessments) has been effective in preparing students to read and trace small programs.