Tracking Changing Learning Goals (Brief Version)

Anne and Steve are building learning goals for CPSC 101. Our efforts fall ***between*** the proposed level for core and non-core courses and focus at the ***lecture*** not ***course*** level. Here, we discuss development of JavaScript (JS) unit goals, illustrating some efforts and benefits of the process.

15 months ago: 101 students were to "learn JavaScript", but detailed goals came only from the text, sample exam questions, and the remarks: "Learning JavaScript will introduce you to programming: the expression of algorithms in a computer executable language!" and "JavaScript will illustrate to you central programming concepts such as: the importance of structuring data and key 'control constructs'." We worked with two examples in lecture and then held an in-class problem session. Students fared poorly on problem sessions and exams, and we were disappointed with our goals' vagueness and tenuous connection to exams.

10 months ago: We articulated our implicit learning goals for the JS unit, connecting to high-level course goals. We added new goals to our existing slides, such as "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 [2 more goals elided]
- be able to modify existing JavaScript code to suit your purposes"

Explicitly articulating goals clarified our own intentions, suggested small changes to the lectures, and eased "hand-off" to TAs and other instructors. Students and staff could also "attribute" quiz and exam questions to explicitly stated goals. But, these goals remain vague. (What does "understand a few key programming 'control constructs' " mean?)

Now: We are again revising goals, with more changes to teaching materials. We previously underemphasized a key skill: reading and tracing existing code. We now explicitly teach this as the "study/model/predict/experiment/refine" process. New goals emphasize this process, e.g.:

- accurately model & predict the behaviour of variables in JS programs
- accurately model & predict the flow of control in a JS program through sequential execution
- 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
- accurately model (and so predict) the flow of control in a JS program through a function call

Lectures pose small programming problems for students to solve, ***before*** detailed instruction. Students discuss solutions and the models implied by each solution, experiment, and refine their models. Assessment is surprisingly straightforward, e.g., we can assess "accurately model & predict the behaviour of variables in JS programs" with a problem like:

```
x = num1;     // Line 1
y = num2;     // Line 2
x = y;        // Line 3
z = x + y;    // Line 4
z = z + 1;    // Line 5
alert("z is: " + z);
```

Sketch the state of each of the variables after each line of code executes. What does the program output?

Formative in-class assessments suggest this approach is working well.