Cognitive Mastery Learning in the ACT Programming Tutor

Albert Corbett

Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
corbett+@cmu.edu

Abstract

The ACT Programming Tutor (APT) is a problem solving environment constructed around an executable cognitive model of the programming knowledge students are acquiring. This cognitive model supports two types of adaptive instructional processes. First, it enables the tutor to follow each student's solution path through complex problem solving spaces, providing appropriate assistance as needed. Second, it enables the tutor to implement cognitive mastery learning in which fine-grained inferences about student knowledge are employed to adapt the problem sequence. This paper outlines the assumptions and procedures of cognitive mastery learning and describes evidence of its success in promoting student achievement. The paper also explores the limits of cognitive mastery as implemented in APT and begins to examine new directions.

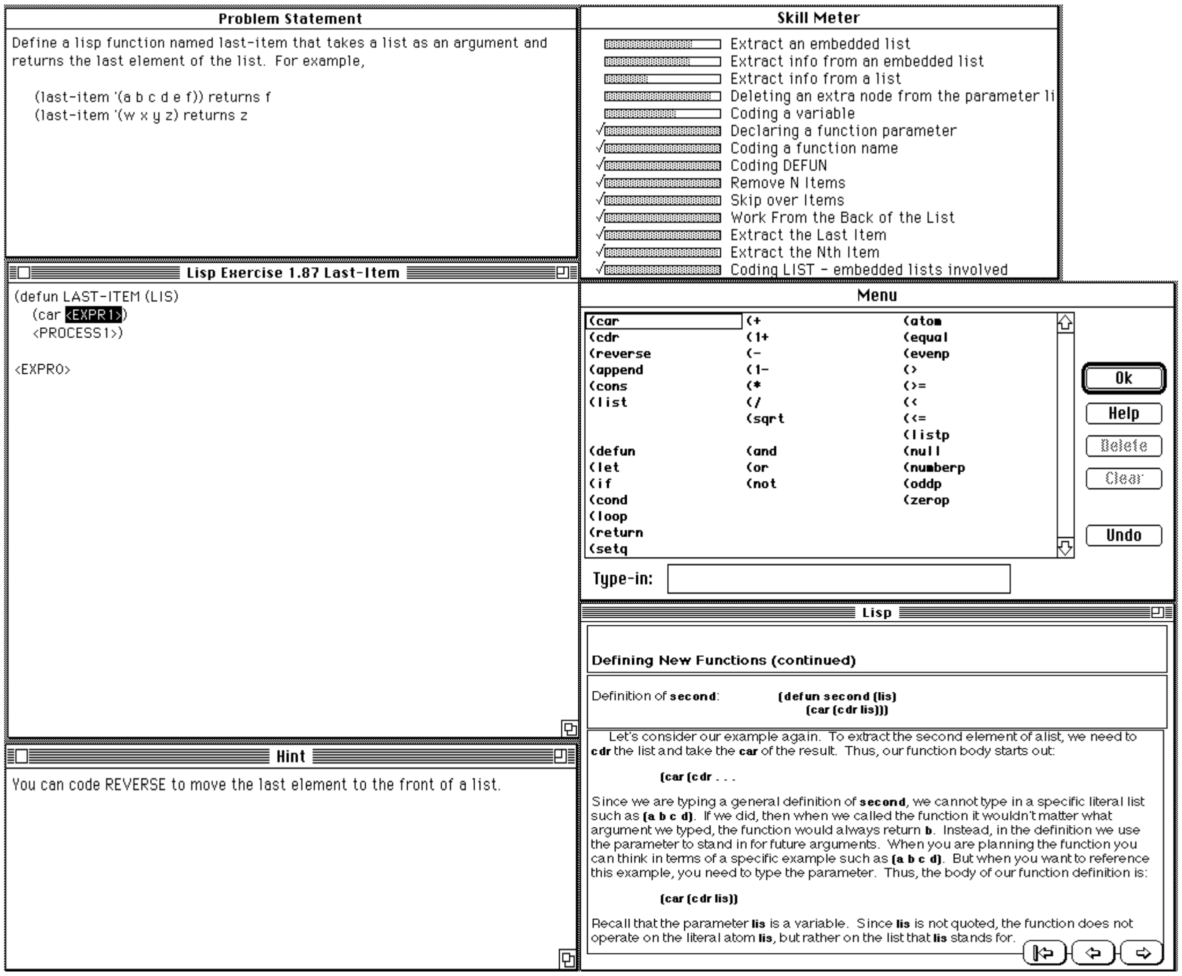
Introduction

Mastery learning was proposed more than thirty years ago as an early form of adaptive curriculum sequencing (Carroll, 1963). Mastery learning has two guiding assumptions. First, that domain knowledge can be decomposed into a hierarchy of component skills. Second, that students should be allowed to progress at their own rate through this knowledge hierarchy, mastering prerequisite knowledge before tackling higher level knowledge. Mastery learning lends itself most readily to individualized instruction (Keller, 1968), but the fundamental principles were also adapted to whole-class instruction (Bloom, 1968). Meta-analyses confirm that mastery learning yields higher mean achievement levels (Guskey and Pigott, 1988; Kulik, Kulik and Bangert-Drowns, 1990; Kulik, Kulik and Cohen, 1979) although the impact is smaller than anticipated (Resnick, 1977; Slavin, 1987) and the underlying assumptions concerning knowledge decomposition are controversial (Resnick and Resnick, 1992; Shepard, 1991).

Over the past decade we have been developing an educational technology at Carnegie Mellon called *cognitive tutors* that brings artificial intelligence to bear on the goals of mastery learning. Each cognitive tutor is constructed around an executable cognitive model of the domain knowledge students are acquiring. As the student works, these tutors monitor the student's growing knowledge state and adapt the sequence of problem solving tasks accordingly. This paper examines cognitive mastery in the ACT Programming Tutor. The underlying cognitive model of programming knowledge is briefly described along with the simple learning and performance assumptions that guide cognitive mastery learning. The paper reviews empirical evidence that cognitive mastery promotes efficient learning, describes some empirical data on the limitations of the process and concludes with a brief discussion of new directions.

The ACT Programming Tutor (APT)

APT is a problem solving environment in which students learn to write short programs in Lisp, Pascal or Prolog. We have employed these three tutor modules to teach a self-paced introductory programming course at Carnegie Mellon since 1984. Figure 1 displays the tutor interface midway through a Lisp exercise. The student has read on-line text in the window at the lower right and is now completing a corresponding set of problems. The current problem statement appears in the window at the upper right and the student's solution appears in the code window immediately below. The interface is similar to a structure editor. The student selects operator templates and types identifiers and constants in the user action window to the right of the code window. In this example, the student has selected the *defun* template to define a new Lisp function, has typed the function name, *last-item*, declared a parameter variable *lis* and coded the first operator in the function body, *car*. The three angle-bracket symbols in the figure, *<EXPR1>*, *<PROCESS1>* and *<EXPR0>* are placeholders which the student will either replace with additional Lisp code or delete. Communications from the tutor appear in the Hint window in the lower left. In this figure the student has asked for a hint on how to proceed and the tutor has provided advice on an operator to accomplish the student's current goal.


Problem Statement
Skill Meter
✓ Extract an embedded list
✓ Extract info from an embedded list
✓ Extract info from a list
✓ Deleting an extra node from the parameter li
✓ Coding a variable
✓ Declaring a function parameter
✓ Coding a function name
✓ Coding DEFUN
✓ Remove N Items
✓ Skip over Items
✓ Work From the Back of the List
✓ Extract the Last Item
✓ Extract the Nth Item
✓ Coding LIST - embedded lists involved
Lisp Exercise 1.87 Last-Item
Menu
(car (cdr (reverse (append (cons (list
(+ (1+ (- (1- (* (/ (sqrt
(atom (equal (evenp (> (>= (< (<= (listp
(defun (let (if (cond (loop (return (setq
(and (or (not
(null (numberp (oddp (zerop
Ok
Help
Delete
Clear
Undo
Type-in:
Hint
Lisp

Define a lisp function named last-item that takes a list as an argument and returns the last element of the list. For example,

```
    (last-item '(a b c d e f)) returns f
    (last-item '(w x y z) returns z
```

```
(defun LAST-ITEM (LIS)
   (car <EXPR1>)
   <PROCESS1>)

<EXPR0>
```

You can code REVERSE to move the last element to the front of a list.

Defining New Functions (continued)

Definition of **second**: **(defun second (lis)
(car (cdr lis)))**

Let's consider our example again. To extract the second element of a list, we need to **cdr** the list and take the **car** of the result. Thus, our function body starts out:

(car (cdr . . .

Since we are typing a general definition of **second**, we cannot type in a specific literal list such as **(a b c d)**. If we did, then when we called the function it wouldn't matter what argument we typed, the function would always return **b**. Instead, in the definition we use the parameter to stand in for future arguments. When you are planning the function you can think in terms of a specific example such as **(a b c d)**. But when you want to reference this example, you need to type the parameter. Thus, the body of our function definition is:

(car (cdr lis))

Recall that the parameter **lis** is a variable. Since **lis** is not quoted, the function does not operate on the literal atom **lis**, but rather on the list that **lis** stands for.

Figure 1. The APT Lisp Module interface.

The Cognitive Model

APT is constructed around a cognitive model of programming knowledge that reflects the ACT-R theory of skill knowledge (Anderson, 1993). ACT-R assumes that a cognitive skill can be modeled as a set of independent production rules that associate problem states and problem solving goals with problem solving actions and subgoals. For example, a simplified version of the first production that fires in coding the Lisp program in Figure 1 would be:

```
If the goal is to define a lisp function
Then  code the operator defun,
      set a goal to code the function name,
      set a goal to code the parameter list,
      set a goal to code the function body.
```

The three productions that fire in coding the body of the function might be:

```
If the goal is to return the last element of a list
Then  code the function car,
      set a goal to move the last element
          to the front of the list.

If the goal is to move the last element of the list
        to the front of the list
Then  code the function reverse,
      set a goal to code the list.

If the goal is to code a list,
   and the list is bound to a variable
Then  code the variable.
```

The APT Lisp module is constructed around a set of several hundred language-specific rules for writing programs called the *ideal student model*. This cognitive model is used to follow the student's solution path through the problem solving space, interpreting each problem solving action action and providing assistance in a process called *model tracing*. The model is also used to monitor the student's growing knowledge state in a process we call *knowledge tracing*.

Model Tracing

In model tracing the cognitive model is employed to follow the student's individual solution path through a complex problem solving space, interpreting each problem solving action and responding appropriately. In each problem solving cycle the student selects a goal node and performs a problem solving action. The tutor applies its problem solving knowledge to the same goal and generates one or more applicable rules that satisfy the goal. If the student's action matches a production rule action it is assumed the student has fired the same cognitive rule. As a result, the production rule is fired and both the internal problem representation and tutor interface are updated. If not, the student action is not accepted. At any time the student can select a goal node and ask for help, in which case an applicable rule is identified and associated help messages provided. Three levels of help are generally available: A reminder of the current goal, general advice on how to achieve the goal and finally, a description of exactly what problem solving action to perform. In Figure 1, the student has asked for help on the node *<EXPR1>*.

This model tracing process has been shown to speed problem solving by as much as a factor of three and increase achievement levels compared to students completing the same problems on their own (Anderson, Corbett, Koedinger, and Pelletier, 1995).

Knowledge Tracing

In addition to tracing the student's solution path for each problem, the tutor draws inferences about the student's knowledge of each production rule in the cognitive model. Each time the student has an opportunity to apply a production rule in problem solving, the tutor updates its estimate of whether the student knows the rule. This knowledge tracing process assumes a very simple learning model; each rule is either in the learned state or the unlearned state. That is, the student either knows the rule or does not know it. A rule can make the transition from the unlearned to the learned state each time there is an opportunity to apply the rule in problem solving. There is no forgetting in the model, so rules do not make the transition from the learned state back to the unlearned state. The model also makes simple performance assumptions. Performance in applying a rule is determined by its learning state, but only probabilistically. If a rule is in the learned state there is a small chance the student nevertheless will slip and make a mistake. If the rule is in the unlearned state, there is still some chance the student will correctly guess the correct action to perform.

The following equation is used in knowledge tracing to update the estimate of the student's knowledge state:

$$p(L_n) = p(L_{n-1}|\text{evidence}) + (1 - p(L_{n-1}|\text{evidence})) * p(T) \quad [1]$$

The probability that a rule is in the learned state following the nth opportunity to apply the rule, $p(L_n)$, is the sum of two probabilities: (1) the posterior probability that the ideal rule was already in the learned state contingent on the evidence (whether or not the nth action is correct) and (2) the probability that the rule will make the transition to the learned state if it is not already there. We use a Bayesian inference scheme to estimate the posterior probability that the rule is already in the learned state $p(L_{n-1}|\text{evidence})$. Following Atkinson (1972) the probability p(T) of a transition from the unlearned to the learned state during problem solving is independent of whether the student applies the rule correctly or incorrectly.

Knowledge tracing employs two learning parameters and two performance parameters, as displayed in Figure 2. The tutor is seeded with a set of best fitting empirical estimates of these four probabilities for each rule. These estimates reflect average performance for the student population. Individual differences among students are also incorporated into the model in the form of four weights per student, one for each of the four parameter types, wL_0, wT, wG and wS. In adjusting the model for a student, each of the four probability parameters for each rule is converted to odds form (p/(1-p)), multiplied by the corresponding student-specific weight and the resulting odds are converted back to a probability. As the student works through the tutor curriculum, these four weights are re-estimated in each curriculum section, by means of regression equations based on raw cumulative error rates and the production rule parameters are readjusted accordingly.

Cognitive Mastery and the Skill Meter

Knowledge tracing is employed to implement cognitive mastery learning in APT. Each lesson in the tutor is divided into sections that introduce a set of related programming production rules. The student reads corresponding text that describes the section topic, then completes a fixed set of required programming problems. This set is typically sufficient to provide at least two opportunities to apply each programming rule introduced in the section. Following this fixed set of problems, individual difference weights are estimated based on the student's cumulative errors across sections and the production rule learning and performance parameters adjusted accordingly. The tutor then presents remedial problem in the section until the probability that the student has learned each rule in the section has reached a criterion value, typically 0.95. As long as any rules remain below this mastery criterion, the tutor will select a problem from among remaining problems whose solution contains the highest proportion of "sub-mastery" productions.

$p(L_0)$	Initial Learning	the probability a rule is in the learned state prior to the first opportunity to apply the rule
p(T)	Acquisition	the probability a rule will move from the unlearned to the learned state at each opportunity to apply the rule
p(G)	Guess	the probability a student will guess correctly if a rule is in the unlearned state
p(S)	Slip	the probability a student will slip (make a mistake) if a rule is in the learned state

Figure 2. The learning and performance parameters in knowledge tracing.

The Skill Meter, as shown in the upper right corner of Figure 1, displays the tutor's estimate of the student's current knowledge state. Each entry represents a production rule in the underlying model and the shaded histogram displays the tutor's estimate of the probability that the student has learned the rule. When this probability reaches the mastery criterion, the rule is checked off. In cognitive mastery, students complete problems in each curriculum section until all the rules are checked.

Empirical Evaluations of Knowledge Tracing

Knowledge tracing and cognitive mastery learning have been evaluated in a series of experiments. In each of these studies students work through the first lesson in the APT Lisp curriculum, much as students do in our self-paced programming course. As described earlier, the lesson is divided into sections that essentially introduce a set of related productions. In each section the student reads a section of text, then completes a fixed set of programming problems with the tutor. Students in cognitive mastery conditions then continue completing remedial problems until mastering all the rules introduced in the section, i.e., until p(L) has reached the criterion value (typically 0.95) for each production in the set. Students also complete one or more quizzes during a study. Each quiz requires the students to complete additional programming exercises similar to the tutor exercises, but with no tutorial assistance.

The following sections summarize empirical results concerning several aspects of cognitive mastery learning: (1) adaptivity to students and impact on test performance, (2) external validity in predicting test performance, and (3) some evidence of the limits of cognitive mastery learning.

Adaptivity: Practical Impact on Problem Solving and Test performance

To examine the practical consequences of cognitive modeling, we have measured both the amount of remedial problem solving and the impact on test performance.

Remedial Problem Solving. The cognitive mastery procedure is fairly reactive to differences in student problem solving performance in the tutor. Across three studies with an average of 28 required problems, students in the cognitive mastery condition averaged 17.33 remedial problems, or about 64% more problems than the minimum required set. The minimum number of remedial problems required by any students averaged 0.33 across the three studies and the maximum number of remedial problems required by any student averaged 62.33. Total time to complete the required and remedial problems in these studies averaged 53.6 minutes. The minimum time required by any student averaged 22.6 across the three studies and the maximum time averaged 132.7 minutes.

Test Performance Comparison. We initially examined the impact of cognitive mastery learning on test performance in the context of a full-semester self-paced programming course. Half the students in the class completed a fixed set of problems with the tutor while the other half worked to cognitive mastery. Students in the baseline condition averaged 87% correct across six quizzes while students in the cognitive mastery condition averaged 95% correct. On the final exam students in the baseline condition scored 72% correct while students in the cognitive mastery condition scored 85% correct. The first effect was significant and the second marginally significant.

We were able to conduct the same comparison for the final cumulative quiz in two of our lab studies. In one study the students in the baseline (fixed curriculum) condition scored 55% correct on the quiz and students in the cognitive mastery condition scored 66%. In the second study students in the baseline condition scored 68% correct while students in the cognitive mastery condition scored 81% correct. These effects are both reliable. In summary, across these four comparisons, students in the cognitive mastery condition are averaging about a letter grade better on tests than students in the baseline fixed curriculum condition.

Learning Effort Comparison. We have directly comparable learning time data for the fixed curriculum and cognitive mastery conditions in two of our laboratory studies. Across these two studies students in the fixed curriculum condition completed 27 tutor problems on average, while the cognitive mastery students completed an average of 43 total problems. Students in the baseline condition spent an average of 41.2 minutes completing their problems while students in the cognitive mastery condition spent 49.0 minutes. Thus, while students in the cognitive mastery condition completed 59% more problems than students in the baseline condition, they only spent 19% more time completing the tutor lesson. This suggests that students are adapting to the tutor at the same time the tutor is adapting to students. Students in the cognitive mastery condition who have an unknown and theoretically unlimited number of problems to complete may be hurrying through the problems more than the students in the fixed curriculum condition.

External Validity: Predicting Test Performance

In addition to examining the practical impact of knowledge tracing, we have empirically validated the predictive validity of the modeling process. Students receive immediate feedback at each problem solving step in the tutor, so we can assume the student is always on a recognizable solution path. Under these conditions we can predict the probability of a correct action at each step with the equation:

$$p(C_{is}) = p(L_{rs}) * (1 - p(S_r)) + (1 - p(L_{rs})) * p(G_r) \qquad [2]$$

That is, $p(C_{is})$, the probability that a student s will perform a correct action at goal i is the sum of two products: (1) the probability that an appropriate rule r is in the learned state for student s times the probability of a correct response if the rule is in the learned state ($p(S_r)$ is the slip parameter in Figure 2), and (2) the probability that an appropriate rule r is not in the learned state for student s times the probability of a correct guess if the rule is not in the learned state ($p(G_r)$ is the guess parameter in Figure 2). Students may deviate from recognizable solution paths in the test environment, but we can compute the probability of completing an entire test problem correctly with the expression:

$$\Pi p(C_{is}) \qquad [3]$$

the product of the probabilities that the student will respond correctly at each successive step in the problem. We have completed two assessments of the predictive validity of the knowledge tracing model and it has proven quite successful (Corbett and Anderson, 1995; Corbett and Knapp, 1996). In the first study the model slightly overestimated average student test accuracy (86% predicted test vs 81% actual performance) and the correlation across students of actual and predicted scores was 0.66. In the second study the model again slightly overestimated average test accuracy (71% predicted vs 62% actual) and the correlation across students of actual and predicted scores was 0.74.

Cognitive Mastery: Summary of Successes and Evidence of Limitations

In summary, cognitive mastery has proven successful by several measures. Average posttest scores are reliably higher for students who work to cognitive mastery than for students who complete a fixed set of required tutor problems. In addition, up to twice as many cognitive mastery students reach "A" level performance (defined as 90% correct) on tests as students in the baseline fixed condition. While time on task is greater in the cognitive mastery than baseline condition, the individualized remediation of cognitive mastery nevertheless reduces total time invested by a group of students in reaching mastery. (The number of remedial problems varies substantially across students and, in the absence of such individualization, mastery could only be ensured if all students complete the greatest number of problems any student needs). Finally, the knowledge tracing model that guides mastery learning in the tutor reliably predicts individual differences in posttest performance across students.

Despite these successes there are some shortcomings in cognitive mastery learning. First, a substantial proportion of students in the cognitive mastery condition fail to reach mastery or "A" level performance. Second, there is typically an inverse correlation between (a) the number of remedial problems students require to reach mastery in the tutor and (b) student's test accuracy. The students who struggle the most in the tutor lesson and require more remedial problems to reach mastery tend to perform worse on the test. Finally, the tutor's knowledge tracing model slightly but systematically overestimates average test performance and empirical evidence indicates that the magnitude of this overprediction is strongly correlated with the depth of each student’s domain knowledge (Corbett and Knapp, 1996; Corbett and Bhatnagar, 1997). These shortcomings provide converging evidence that there are diminishing education benefits of simply providing more and more remedial problems.

Future Directions

More recently, we have begun to develop “perspicuous” problem solving interfaces that are intended improve learning efficiency by helping students make more sense of problem solving as they work. We have introduced two instructional interventions in the Lisp Programming tutor that have very difference impacts on learning, (a) animated graphical feedback, designed to help students understand subtle representational distinctions and (b) subgoal scaffolding, designed to help students formulate plans. Plan scaffolding dramatically reduces learning time without changing test performance, while augmented feedback substantially increases test accuracy without affecting learning time (Corbett and Trask, in press). Two current questions are how to generalize these interface modifications and whether we can further enhance their effectiveness by making them adaptively available to some students and not others.

Acknowledgements

This research was supported by the Office of Naval Research grant number N00014-95-1-0847 and by NSF grant number 9720359 to CIRCLE: Center for Interdisciplinary Research on Constructive Learning Environments

References

Anderson, J. R. 1993. *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R.; Corbett, A.T.; Koedinger, K.R.; and Pelletier, R. 1995. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences.* , 4:167-207.

Atkinson, R.C. 1972. Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology*, 96:124-129.

Bloom, B. S. 1968. Learning for mastery. In *Evaluation Comment, 1*. Los Angeles: UCLA Center for the Study of Evaluation of Instructional Programs.

Carroll, J. B. 1963. A model of school learning. *Teachers College Record*, 64:723-733.

Corbett, A.T.; and Anderson, J.R. 1995. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction,* 4:253-278.

Corbett, A.T.; and Bhatnagar, A. 1997. Student modeling in the ACT Programming Tutor: Adjusting a procedural learning model with declarative knowledge. *Proceedings of the Sixth International Conference on User Modeling*. New York: Springer-Verlag Wein.

Corbett, A.T.; and Knapp, S. 1996. Plan scaffolding: Impact on the process and product of learning. In C. Frasson, G. Gauthier, & A. Lesgold, (Eds.) *Intelligent tutoring systems: Third international conference , ITS ‘96.* New York: Springer.

Corbett, A.T.; and Trask, H.J. in press. Instructional Interventions in Computer-Based Tutoring: Differential Impact on Learning Time and Accuracy. *Proceedings of ACM CHI’96 Conference on Human Factors in Computing Systems.*

Guskey, T. R.; and Pigott, T.D. 1988. Research on group-based mastery learning programs: A meta-analysis. *Journal of educational resear*ch, 81:197-216.

Keller, F. S. 1968. "Good-bye teacher...." *Journal of applied behavioral analysis*, 1:79-89.

Kulik, C. C.; Kulik, J.A.; and Bangert-Drowns, R.L. 1990. Effectiveness of mastery learning programs: A meta-analysis. *Review of Educational Research*, 60:265-299.

Kulik, J. A.; Kulik, C.C.; and Cohen, P.A. 1979. A meta-analysis of outcomes studies of Keller's Personalized System of Instruction. *American Psychologist*, 34:307-318.

Resnick, L. B. 1977. Assuming that everyone can learn everything, will some learn less? *School Review*, 85:445-452.

Resnick, L. B.; and Resnick, D.P. 1992. Assessing the thinking curriculum: New tools for educational reform. In B. Gifford & M. O'Connor (eds.) *Changing assessments: Alternative views of aptitude, achievement and instruction*. Boston: Kluwer.

Shepard, L. A. 1991. Psychometrician's beliefs about learning. *Educational Researcher*, 20:2-16.

Slavin, R. E. 1987. Mastery learning reconsidered. *Review of educational research*, 57:175-213.