

Learning Curve Analysis for Programming: Which Concepts do Students Struggle With?

Kelly Rivers, Erik Harpstead, and Ken Koedinger

Educational Data Mining for Programming

- ITiCSE working group paper on EDM and Learning Analytics in Programming
 - State of the art- focused on simple metric analysis, not so much on learning of content in code
- Plenty of approaches have been developed in the more general fields of learning science, like intelligent tutoring systems and educational data mining
 - Defining *knowledge components* as the concepts students exercise while working
 - Investigating how students learn knowledge components *over time* using learning curves
 - Highlighting *potential shortcomings* of instructional interventions

Research Goal

How can we apply knowledge component modeling and learning curve analysis directly to open programming tasks?

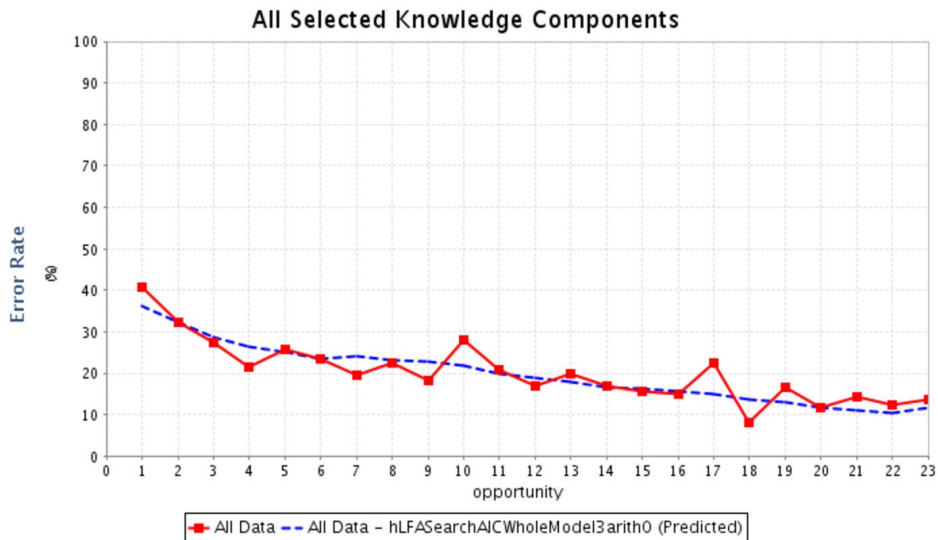
Background

What is a Knowledge Component (KC)?

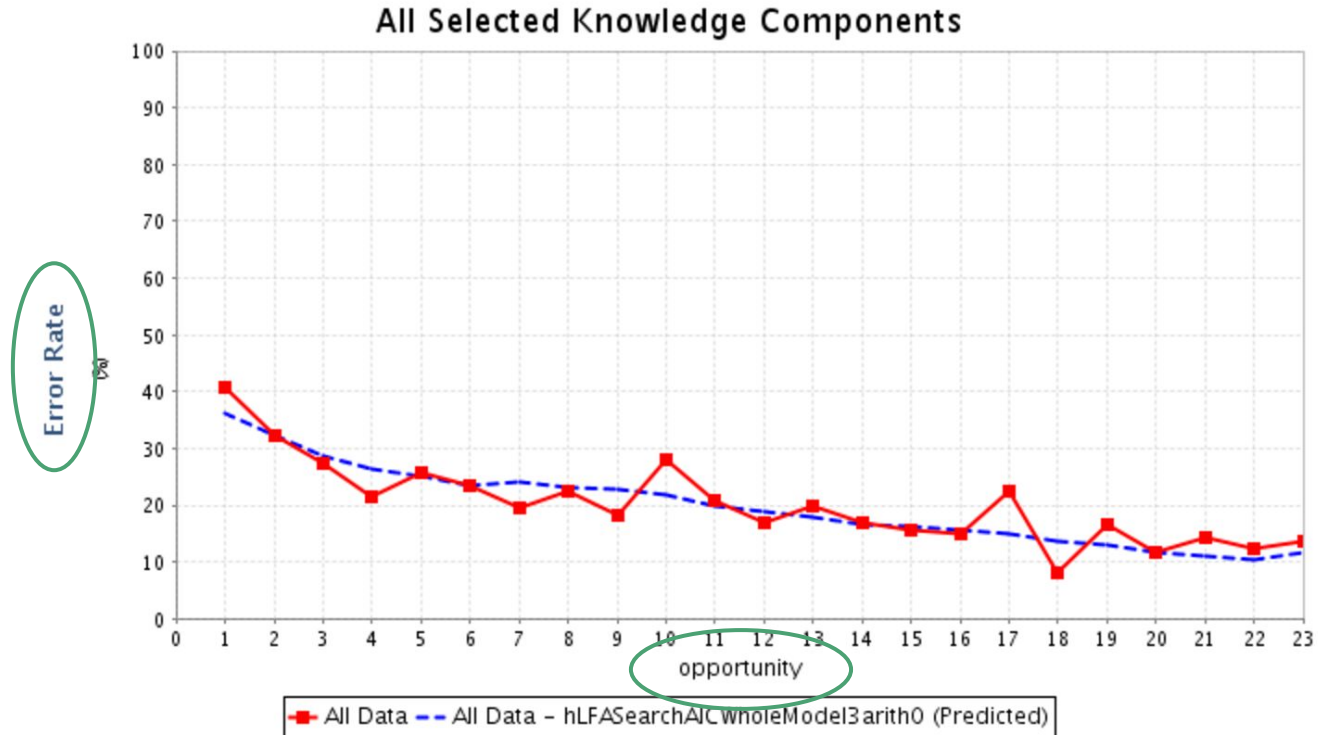
- A Knowledge Component (KC) is “...an acquired unit of cognitive function or structure that can be inferred from performance on a set of related tasks”
(Koedinger, Corbett, & Perfetti, 2012)
- A unifying formalism for things like skill, concept, principle, fact, schema, production rule, *thing-to-be-learned*, etc.
- Math example: finding the area of a circle

Learning Curve Analysis (LCA)

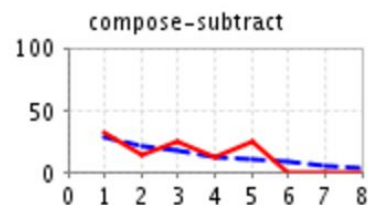
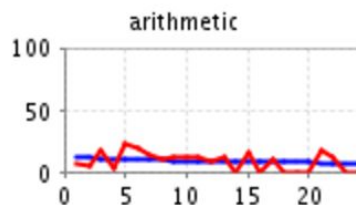
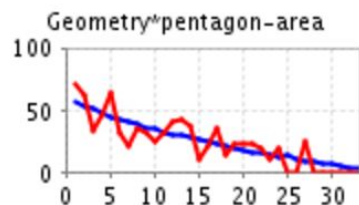
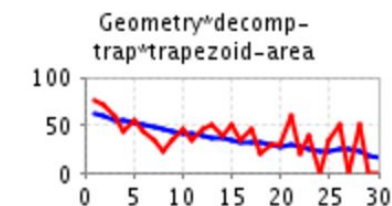
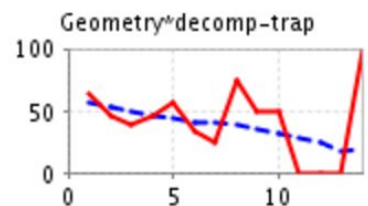
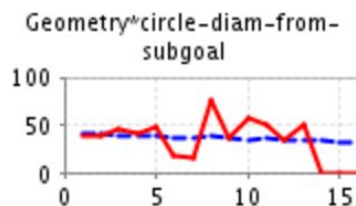
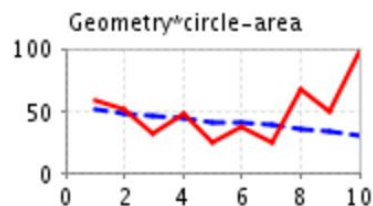
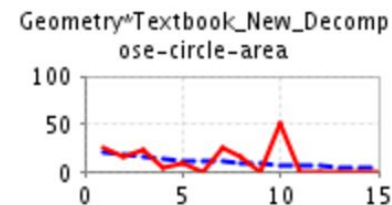
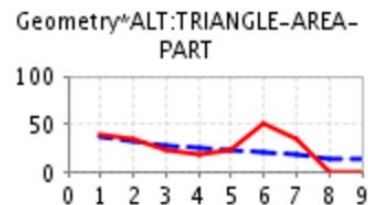
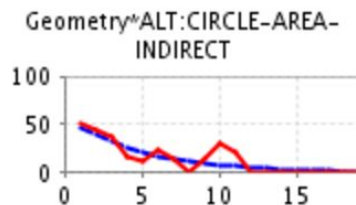
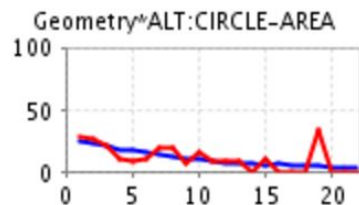
- First created to analyze production rules in the LISP Tutor (Anderson, Conrad & Corbett, 1989)
- Additive Factors Model (AFM)- developed to evaluate cognitive models by statistically fitting them to data (Cen, Koedinger & Junker, 2006)
- Now used in Datashop (pslcdatashop.web.cmu.edu) to provide visual feedback and exploratory data analysis to tutor developers (Koedinger et al, 2010)



Learning Curves



KC Learning Curves



Additive Factors Model (AFM)

- AFM is a special form of mixed effects logistic regression for modeling student performance over time which estimates:
 - Individual student intercepts (capturing initial ability)
 - KC intercepts (capturing relative difficulty of each KC)
 - KC slopes (capturing general learning rate of each KC)
- Fit estimates from AFM can be used to plot learning curves against the actual student performance data and categorize different learning patterns

Questions for Applying LCA

- What are the
 - **KCs**
 - **Opportunities**
 - **Error Rates**
- of programming in an open editor?

Methodology

What are the **KCs** of Programming?

- Performance: test cases/constraints?
 - Not transferrable across problems
- Cognitive function: algorithm/plan implementation?
 - Difficult to define the opportunities for these KCs
 - Though it has been done before...

Previous Work: The LISP Tutor

- Heavily structured editor for tutoring LISP
 - Immediate feedback after every token written
- Authored 500 production rule KCs that determined when to apply specific code tokens

Production Rule: p-insert

IF the goal is to insert one element into a list

THEN code cons and set subgoals:

To code the element

To code the list

Define a function called "create-list" that accepts one argument, which must be a positive integer. This function returns a list of all the integers between 1 and the value of the argument, in ascending order. For example,

(create-list 8) returns (1 2 3 4 5 6 7 8).

You should count down in this function, so that you can just insert each new number into the front of the result variable.

CODE for create-list

```
(defun create-list <parameters>
  <process>)
```

Source: Anderson, Conrad, & Corbett, 1989

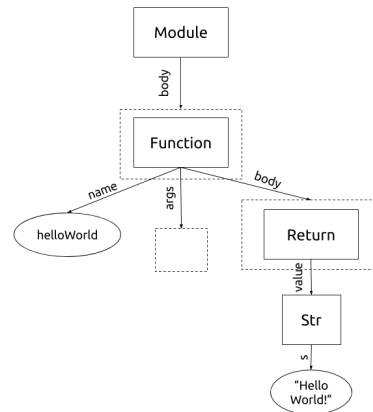
What are the **KCs** of Programming?

- Program tokens - AST nodes?
 - Easily modeled by matching tokens to underlying program representation
 - AST- abstract syntax tree
 - Can be used as a starting representation of algorithmic concepts

CODE

```
def helloWorld():  
    return 'Hello World!'
```

AST



What are the **Opportunities** of Programming?

- Traditional tutoring systems: every step is executed separately, every step is an opportunity
- A task/problem is composed of multiple **steps**, where each step usually corresponds to one KC
- An **opportunity** occurs the first time a student submits an answer to a step; following attempts are not counted after feedback is given

What are the **Opportunities** of Programming?

- Open coding: all steps in the task are combined into one program, which is modified in an iterative design process
- Our definition: a programming **step** is a deliberate feedback request by the student (running the compiler, running test cases, asking for help)
- Each step is composed of **opportunities** for all of the token KCs that occur in the code. These opportunities are evaluated in parallel.
- **KC Model Test:** First-Attempt-Only vs. All-Attempts
 - Do we count every student submission as an opportunity or only their first attempt at the problem?

How Do We Measure **Correctness** in Programming?

- Whole program correctness: test cases
 - Can't use this for individual KCs, since one incorrect KC will penalize the others
- More syntactically detailed correctness: which tokens change between the current state and a correct state?
 - Use prior work on hint generation to determine what the best *goal state* is, given the *current state* (Rivers & Koedinger, 2014)
 - Then can determine the changed tokens by comparing the two ASTs to find *semantic differences*

How Do We Measure **Correctness** in Programming?

- A token KC is **incorrect** when:
 - Commission error: The token occurs in the edit between the current state and the goal state
 - Omission error: The token is missing from the solution and this is the last attempt the student makes
- A token KC is **correct** when...
 - Do we evaluate if a KC is correct every time a student submits, or only when the relevant code is modified?
 - **KC Model Test:** All-Steps vs. Modified-Steps
 - All-Steps Model: A state is correct when it is not incorrect
 - Modified-Steps Model: A state is correct when the token occurs in the edit between the previous attempt and this attempt, or this is the first attempt

KC Model Formats

	First Attempt	All Attempts
Modified Steps	Traditional tutor approach- only count the first opportunity for each KC	Count every opportunity where the given KC has changed
All Steps	Count the first opportunity in every session	Count every opportunity for every KC

Analysis

Analysis Plan

1. Annotate data with KC labels
2. Upload labeled data to Datashop to fit AFM
3. Using AFM estimates, scrutinize individual KC learning curves for interesting cases

Data

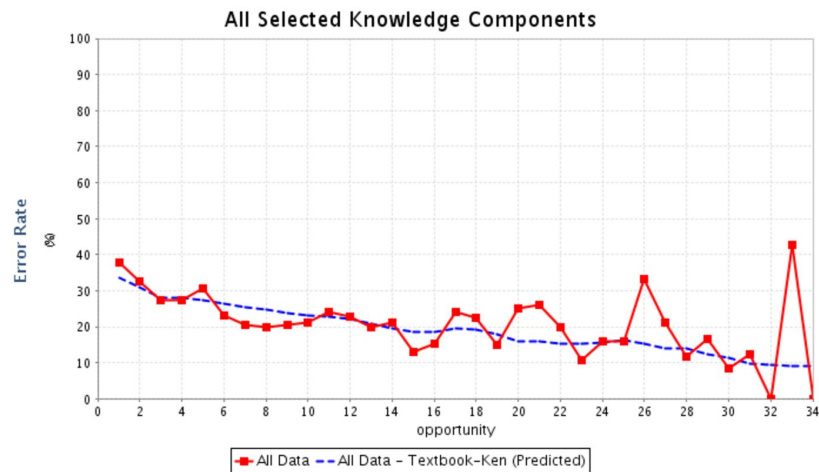
- Study run in Spring 2016 with two Carnegie Mellon University CS1 courses
 - Study on usage of hints, but we're not looking into that just yet
- 40 optional practice problems (ranging from basic expressions to dictionaries and lists)
- 89 students chose to participate, generated 2907 submissions and 380 hint requests

Model Generation

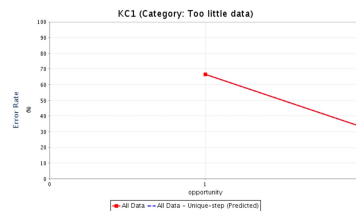
1. Generate a solution space
2. For each problem, find the set of KCs used in the teacher's solution
3. For each submission:
 - a. Use hint generation to get the code to a parseable state (by fixing syntax)
 - b. Identify the closest goal state (using path construction algorithm)
 - c. Use a tree differ to find all edited AST nodes between the state and the goal
 - d. Apply tags according to the model rules

Results- what are we looking for?

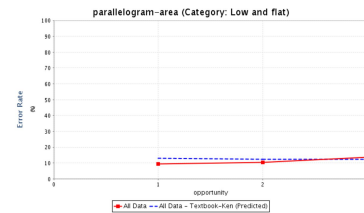
Typical learning curve



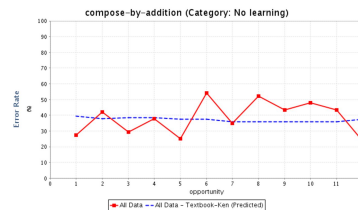
Not Enough Data



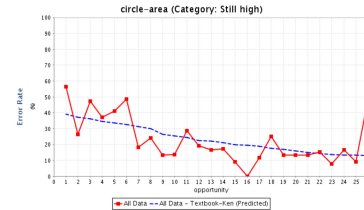
Already Learned



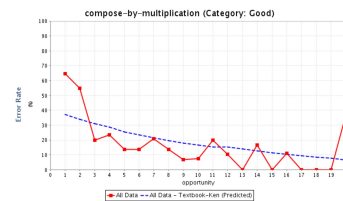
No Learning



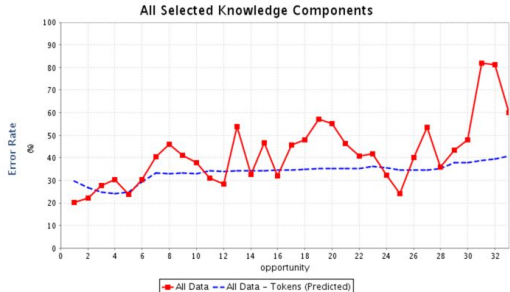
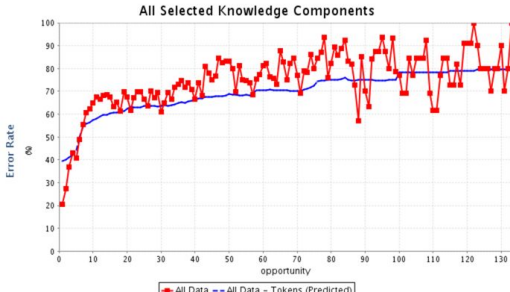
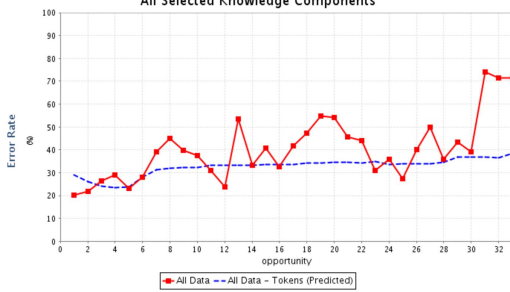
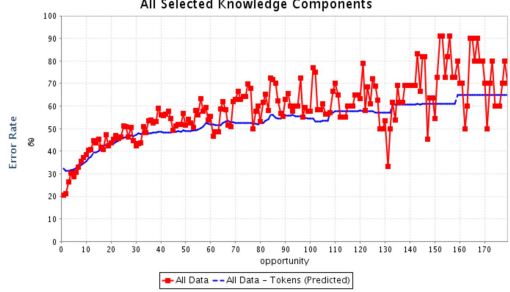
Still Learning



Good Learning



Results - Overview

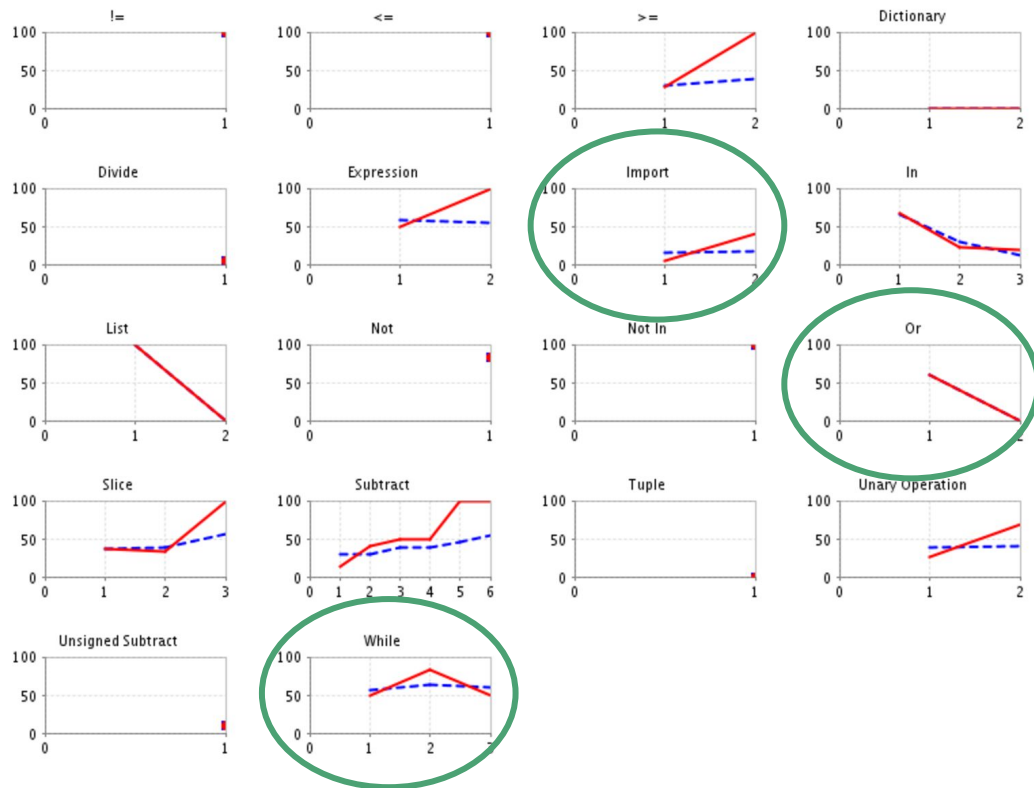
	First Attempt	All Attempts
Modified Steps	<p>All Selected Knowledge Components</p>  <p>Legend: All Data (red line with squares), All Data - Tokens (Predicted) (blue dashed line)</p>	<p>All Selected Knowledge Components</p>  <p>Legend: All Data (red line with squares), All Data - Tokens (Predicted) (blue dashed line)</p>
All Steps	<p>All Selected Knowledge Components</p>  <p>Legend: All Data (red line with squares), All Data - Tokens (Predicted) (blue dashed line)</p>	<p>All Selected Knowledge Components</p>  <p>Legend: All Data (red line with squares), All Data - Tokens (Predicted) (blue dashed line)</p>

Results- KC overview

- **Common Categories:** Too little data, No learning
- **Medium Categories:** Already learned, Good learning
- **Rare Categories:** Still learning

Caution: these can change!

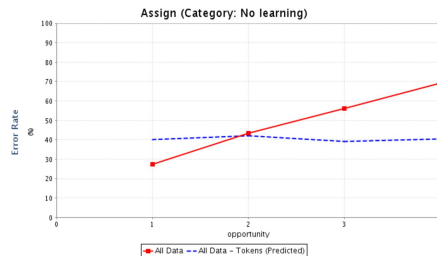
Too Little Data KCs



No Learning KCs

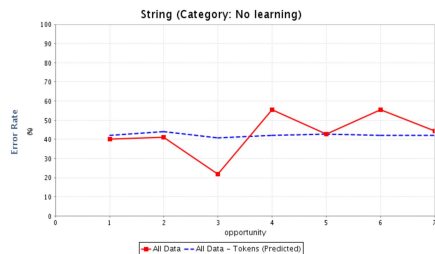
- Backwards Learning

- Assign, Attribute, Power, *, >



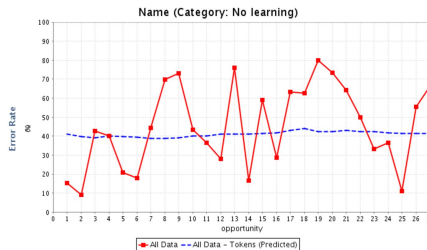
- No Learning

- Compare, Subscript, Index, +, //, %, ==, String



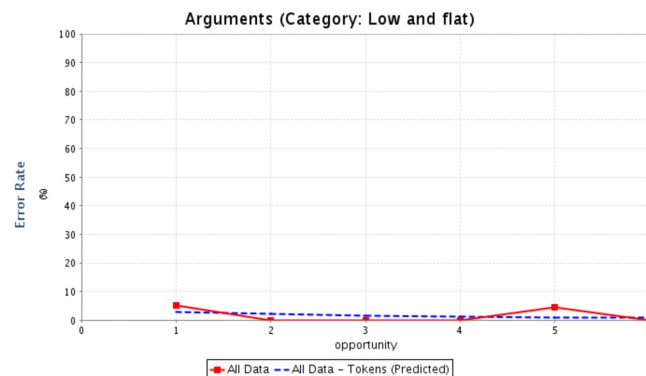
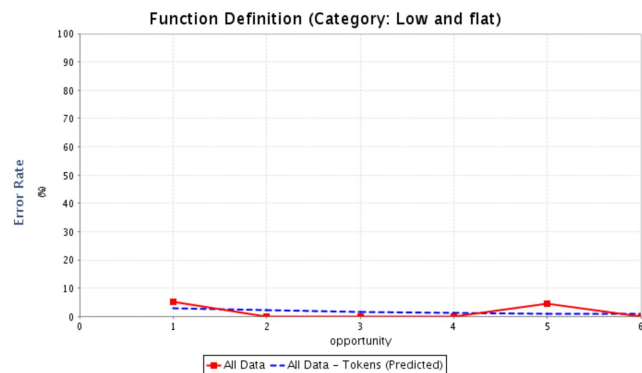
- Spikey Learning Curves

- Return, Binary Operation, Function Call, <
 - Variable Name, Number



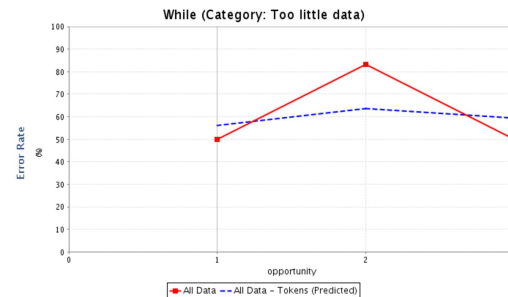
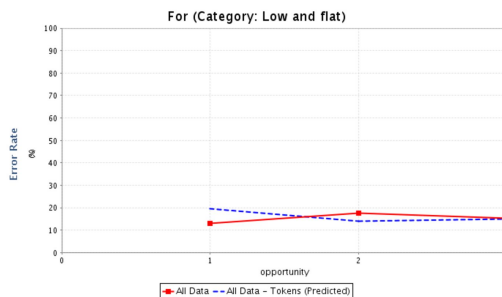
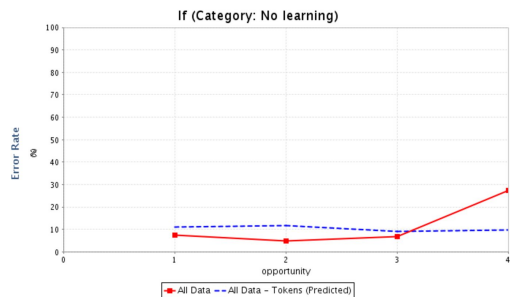
Already Learned: Function Headers...

- Only needed in first 6 problems; later problems provided starter code
 - Possible that all mistakes are being caused by syntax



... and Control Structures?

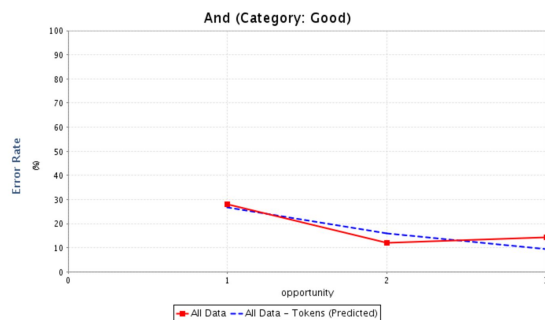
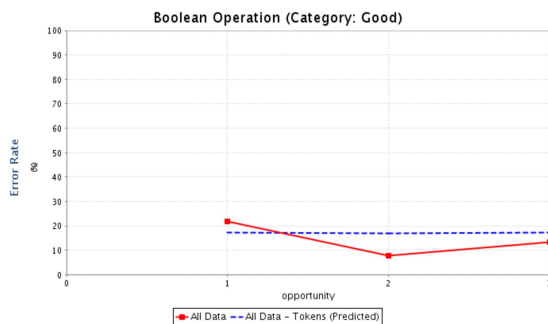
- If and For statements appear to be easy to use
- While loops are harder to evaluate, since we didn't have enough problems!



All Data			
Opportunity Number	1	2	3
Number of Observations	16	6	2

Observed Learning: Boolean Operations

- Students did seem to improve in using *and* expressions
- *Or* expressions not being used as much



Extra! Validation with student intercepts

- Learning curves are evaluated using AFM models
 - These models use different intercepts for the set of KCs and for individual students
- Student intercepts- supposed to model the student's prior learning/ability
 - Can be validated by looking at student's actual exam scores!
- Correlation between student intercepts and student exam scores: **0.377**
 - The model as a whole is moderately correlated with student outcomes

Final Thoughts

Main results

- We envisioned a new way to represent **programming KCs** and **steps**
- We evaluated **learning curve results** for programming data generated in a modern, **unstructured coding** context

Limitations

- Using AFM and learning curves in non-traditional ways
- Changing representations of steps and correctness
- Not using data from a mastery-paradigm system
 - Also, programming KCs are generally wonky
- Currently not counting syntax changes in KC modeling
- Current modeling assumes a very granular view of programming KCs

Alternative Approaches

- Categorize some nodes into larger classes
 - $<$, $>$, $<=$, $>=$, $==$, $!=$ are all one type; in and not in are another
- Use AST node context instead of type
 - Instead of 'Boolean comparison', 'If test'
 - Or combine the two!
- Modify KCs looked at per problem
 - Use student's goal state instead of teacher's
- Modify correctness of KC steps
 - Maybe only highest-level type in an edit counts, not interior nodes
- Probably tons more!
 - What do you think?

Acknowledgements

Erik Harpstead and Ken Koedinger

Jason Imbrogno

This work was supported in part by Graduate Training Grant awarded to Carnegie Mellon University by the Department of Education (# R305B090023).



The Additive Factors Model

$$\ln \frac{p_{ij}}{1 - p_{ij}} = \theta_i + \sum_k \beta_k Q_{kj} + \sum_k Q_{kj} (\gamma_k N_{ik})$$