

Estimating Programming Knowledge with Bayesian Knowledge Tracing

Presentation for ITiCSE'09

Jussi Kasurinen

Uolevi Nikula

Learning programming

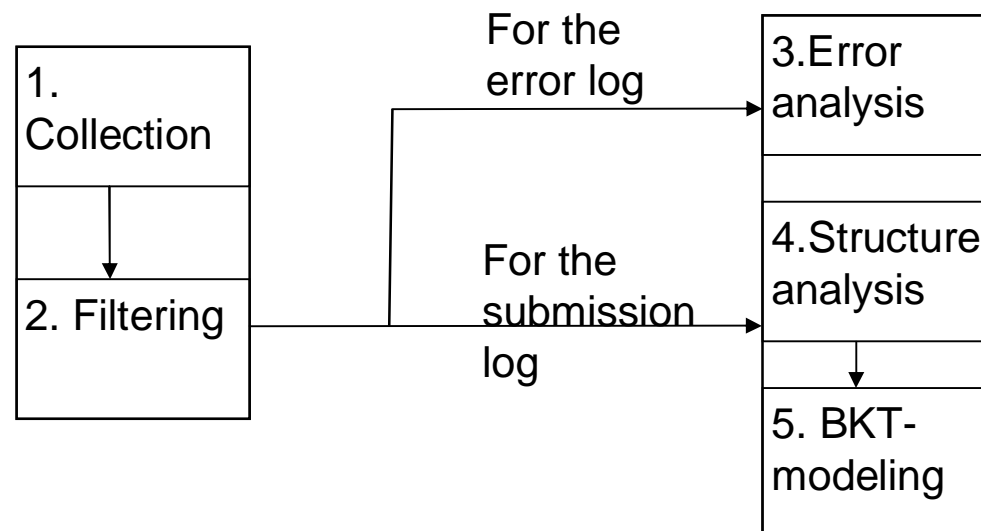
- Programming is hard to learn: students have to learn several concepts and structural rules before they can create anything interesting.
- Programming is learned by first understanding the structures themselves, and later by combining these structures to create more and more complex functionalities.
- Programming languages offer different “building blocks” – structures – of varying complexity to enable this behavior.
 - Iterator, function, datastructures, object, array, class, exception, condition...

Learning programming

- However, there are usually several similar or redundant structures (iteration, condition especially: for/while/do, if/select/case).
 - Some structures can overtake activities (conditions, exception handlers)
- Students seem to start favor some structures over another, to a point where personal preferences supersede practicality.
 - Besides practicality, there are also undesirable issues like shortcuts or disregard of good programming manners.
- However, the way the students program can be used to assess their programming knowledge:
 - What structures are they using?
 - Do they understand all of the structures?
 - Are they using the structures correctly?
 - What kind of errors are they encountering?

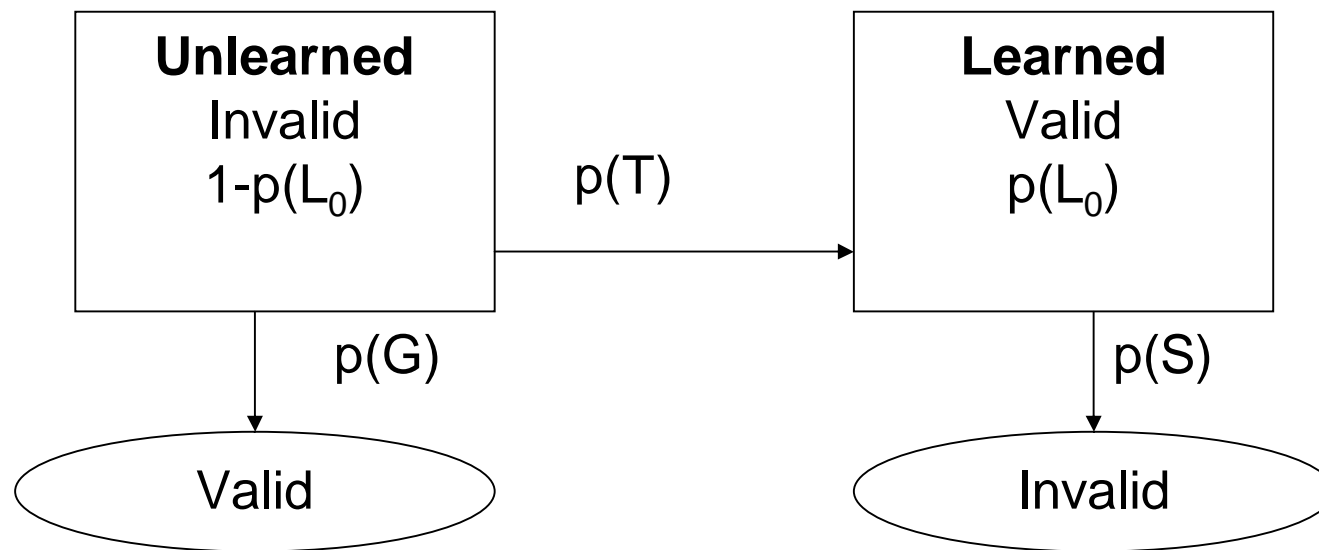
Analysis model

- Error analysis: error type accumulations to observe trends and difficult concepts in programming.
- Structure analysis: what structures were used to implement source codes?
- Bayesian knowledge tracing (BKT)-modeling: probability of learning based on the applied structures.



Bayesian Knowledge Tracer

- $p(L_n)$, prob. of having been learned, $L_0 = 0.35$ ¹
 - n is the amount of iterations
- $p(T)$, prob. to learn by implementing correctly, set to 0.7 ²
- $p(G)$, prob. to guess the right answer. set to 0.3 ²
- $p(S)$, prob. to make mistake while understanding the concept, set to 0.1 ²



¹ Based on student background survey

² Based on literature

Bayesian formula

(1) $L_n = p(L_{n-1} | R_n) + (1 - p(L_{n-1} | R_n)) * p(T)$ where

(2) if correctly applied: $p(L_{n-1} | R_n) =$

$$[(1-p(S)) * p(L_{n-1})] /$$

$$[((1-p(S)) * p(L_{n-1})) * (p(G) * (1- p(L_{n-1}))))]$$

(3) if incorrectly applied: $p(L_{n-1} | R_n) =$

$$[p(S) * p(L_{n-1})] /$$

$$[(p(S) * p(L_{n-1})) * ((1 - p(G)) * (1- p(L_{n-1}))))]$$

Example calculations ($L_0 = 0.35$)

- "11001":

$L_n, n=$	1	2	3	4	5
Correct?	1	1	0	0	1
Bayes	0.62	0.96	0.92	0.85	0.98

- "0001":

$L_n, n=$	1	2	3	4
Correct?	0	0	0	1
Bayes	0.13	0.29	0.34	0.92

- In here, correct means either "not deemed necessary and not used" or "necessary and used"; 1 = True, 0 = False.
- Becoming "Learned" requires approximately two successful implementations in a set of five exercises.

Data collection

- We used our collaborators virtual learning environment (VLE) to collect student submissions.
www.viope.com (*in Finnish*)
 - 3732 source codes and 9420 errors
- VLE had a few additional services:
 - Submission validation: the source code worked as intended (input/output matching)
 - Hinting service: “First aid” service to help students in a need of assistance.
 - Plagiarism detection: similarity testing with prior submissions. Structural and literal comparisons possible.
 - Student statistics: teachers could observe how much time students had used, what kind of errors they had encountered, in which time of the week/day they are actively working etc...

Data filtering

- Out of 41 programming assignments, 12 were considered usable.
 - Simple, straightforward assignments from the first weeks were rejected.
 - Assignments with learning tool were considered too refined because of pseudo code and detailed descriptions.
- Final 12 were from the latter parts of the course, required 40-50 lines of Python code and enabled students to pursue individual strategies.
 - Out of 3732 sources, 675 were finally used.
- Error analysis was done to the whole set;
 - the rationale for this was to observe if there were trends or other phenomena that could explain the possible structure selections for the assignments from the latter parts of the course, as in “high amount of index-errors from iterations à students avoid for-structure which causes them.”

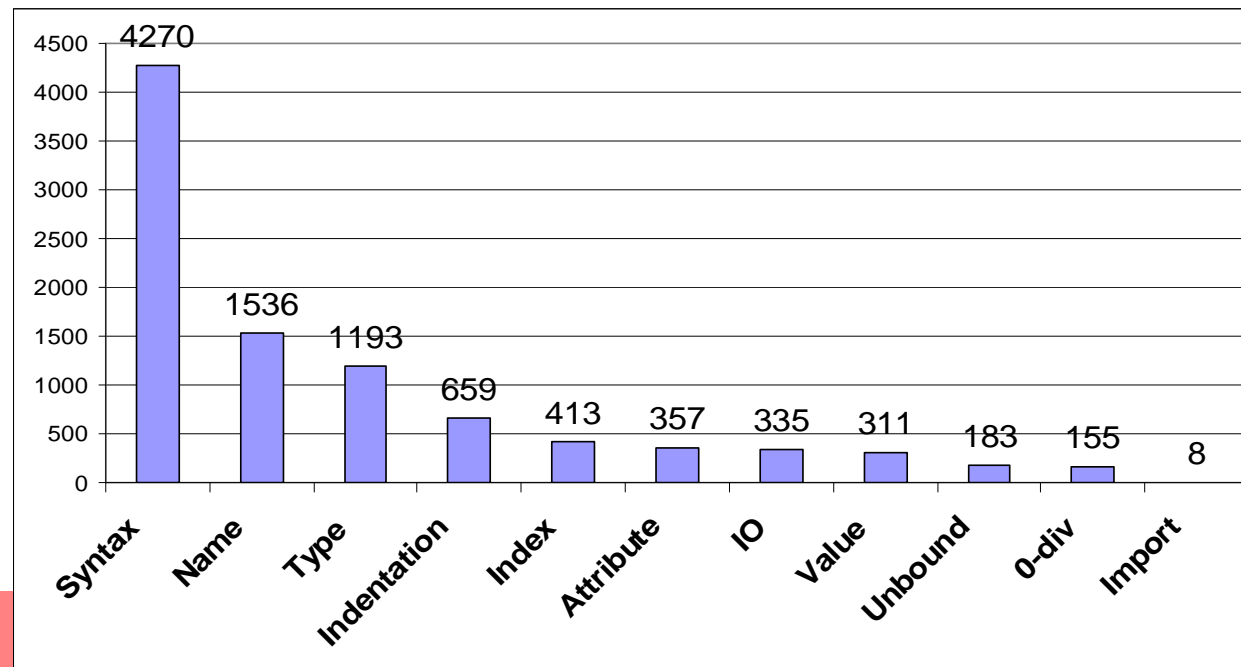
Data filtering

- Structure mapping was done with Python program reading database dump:
 - Search for syntax keywords from non-commented lines.
 - Only functional source codes in the dump!
- Final calculations and BKT-analysis were done with an Excel spreadsheet.

Examples of Excel and Python source

Error analysis

- Three groups of errors:
 - syntax-related (syntax, name, indentation)
 - structure-related (index, attribute, IO, unbound)
 - ambiguous or special (value, type, 0-division, import)

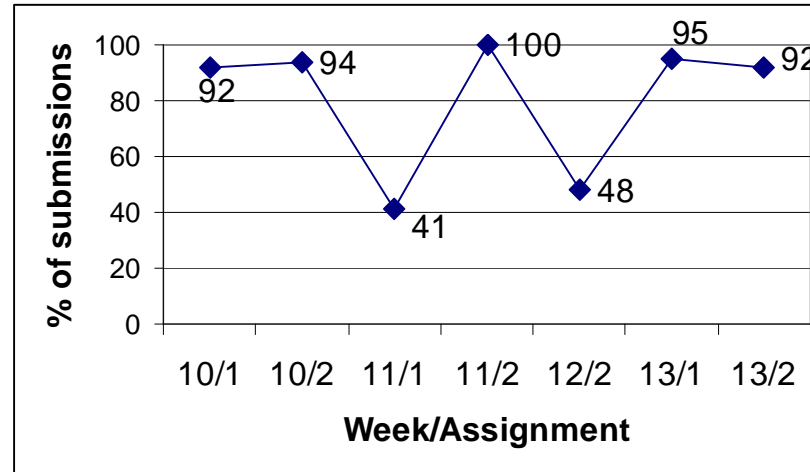


Structure analysis

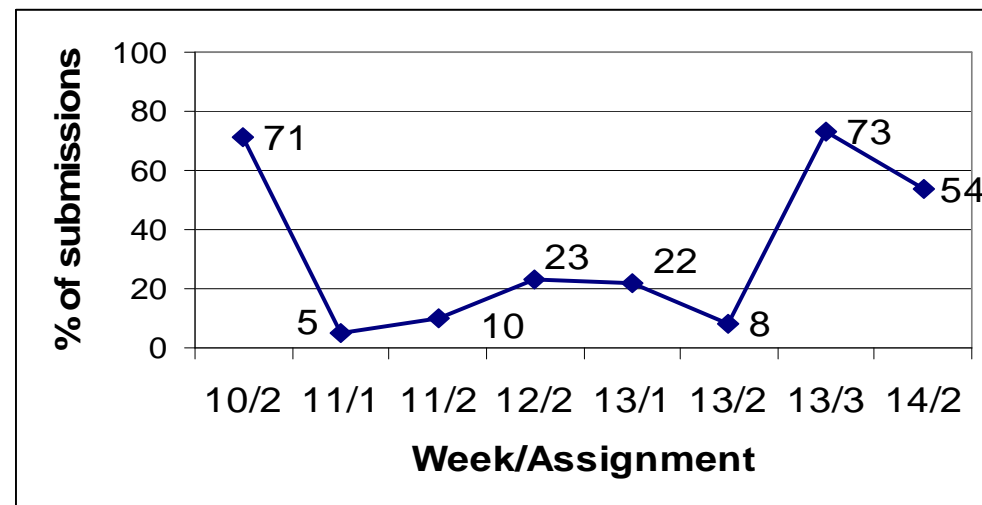
- Five categories were tested:
 - while should be used if the number of iterations is not known, otherwise for.
 - Every input should be taken as a string with `raw_input` and converted to a suitable type.
 - Each open file should be closed after use.
 - Functions (`def`) should be used to implement periodically occurring behavior .
 - Each type conversion to number and file opening for reading should be secured with exception handler.
- As there is only one conditional structure in Python (`if-elif-else`), there was no incentive to test it.

Structure examples

File handling:

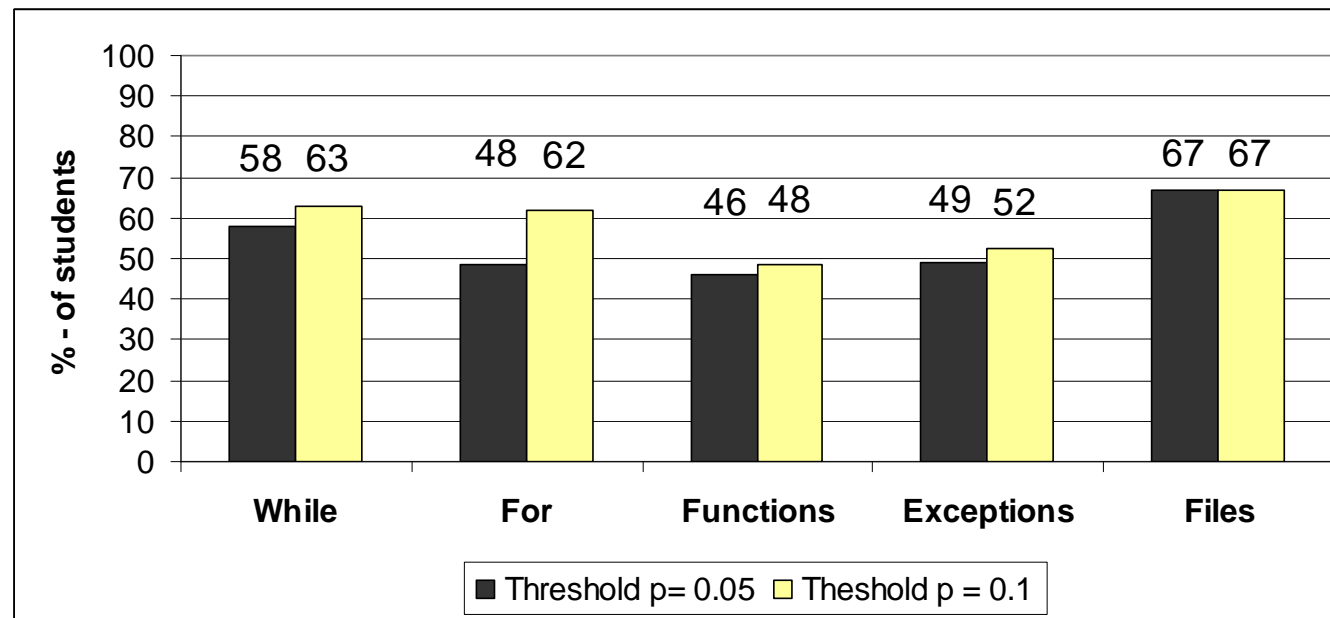


Exception handling:



BKT-analysis

- Results with two thresholds: 95% and 90% confidence for learning the structure:



Limitations

- No support for longitudinal study, no model for forgetting taught topics.
 - ACT-R model for this, but for 14 weeks unnecessarily large; BKT has had “a history of success” in programming and algebra.
- Accuracy of Python interpreter error analysis has not been established.
 - Used only to observe trends.
- No support for individual differences, prior experience not taken into account.
 - Obviously this causes the model to underestimate the gifted and/or experienced students.
- Individual approaches differ, what is “correct” approach?
 - There are only handful of rules on what is “correct” programming, however observing if one construct is applied at all is feasible.

Results

- So what did we find out?
 - Syntax difficulties seem to be the deal breaker: Students have difficulties expressing themselves.
 - Students avoid more complex structures if they can be bypassed with a compilation of easier ones.
 - Roughly 55% of the students actually learned to utilize the programming structures, even if the course passing percentage was over 70%. → one out of five passing students have only limited understanding after the course.
 - Some students are surprisingly capable; new form of iteration made by combining exception handler and recursive function.

Discussion

- ... But we already (kinda) know these things?
- Yes, but for us that means that the proof-of-concept –model is sensible.
- Only little amounts of novel data, but possibility to elaborate the method and develop teaching tool to observe student progress.
 - Also ability to observe technical soundness of the exercises, tools.

Future work

- Further work to validate the model.
- Design exercises to suit more closely to the BKT-analysis.
- Further in the future:
 - Integration of the analysis tool to the existing environment.
 - Tailored assignments to promote individual learning for struggling students.

Thank you for your time

- Here's a picture of our campus area to please our PR department:

