Helping Students Build a Mental Model of Computation

Steve Kollmansberger
South Puget Sound Community College
2011 Mottman Rd SW
Olympia WA 98512
+1 360 596 5307
skollmansberger@spscc.ctc.edu

ABSTRACT
Introductory computer science classes are known for having a high attrition rate. Some authors believe this is due to students' difficulties in establishing a mental model of computation. We present a tool designed to assist students in actively building a mental model of computation, and evaluate its use in an introductory programming class. We show use of the tool leads to 40% increase in course completion and that the students believe the tool helps them understand programming.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms
Algorithms, Design, Experimentation

Keywords
Active learning, program tracing, online tutoring

1. INTRODUCTION AND BACKGROUND
Dehnadi identified that a student's ability to develop a consistent mental model appears important to success in introductory programming courses [5]. Although the exact diagnostic was later disproven [4, 2], the usefulness of a valid mental model remains plausible. Indeed, Reges found that a single AP test question served as an excellent predictor of student success [8]. The reason? The question required students to have a working mental model of computation.

Mental models of computation, in the form of "computational thinking" are increasingly being recognized as important student learning outcomes [3]. This evidence suggests that beyond planning and general problem solving, a consistent mental model of computation is essential for developing programming ability.

ITiCSE'10, June 26–30, 2010, Bikent, Ankara, Turkey.

Although previous work has been devoted to detecting and predicting success using mental models [5, 8], we would like to consider the possibility of assisting students to build a correct mental model. The idea of using a tool to develop early students programming skills is certainly not unique. For example, Jin developed a program planning tool which assists students in discovering the role and purpose of variables, as well as focusing on the step-wise program flow [6]. Likewise, Boisvert developed a tool which animates the process of program development, with an emphasis on "playback" of recorded histories [1]. Vainio and Sajaniemi proposed that students' poor tracing skills contributed to a large group of learning difficulties [10]. Indeed, software which automatically traces a student through a program has been developed [9] and even many IDE debuggers sport this ability.

We developed a tool that emphasizes program tracing in an interactive way that requires the student to actively be involved in the trace, but also ensures that the student cannot become stuck. The tool is set up to allow the student to repeat each trace for a higher score. We find that students using this tracing tool are more likely to complete the introductory programming class and say they find the tool a good use of time and beneficial to their understanding of how a program executes.

2. THE TOOL
We developed the tool to correspond to the topics covered in our introductory class. The tool is extensible and programming language neutral. Each exercise exists as an XML file which is rendered by the tool into the language being used. In our case, Visual Basic.NET 2008 is the language used in the introductory course. The tracing program allows the student to step through each statement that is executed. The student then selects which statement, statement part, or expression will be evaluated next.

The students are required to consider the operation in enough detail to describe it mechanically. For example, when entering a for loop, the student must first select and work through the counter initialization, then they must select and work through the condition, and then they may select and begin the for loop body (See figure 1).

When expressions are selected, students "drill-down" piece by piece and then build the expression back up (See figure 2). This "drill-down" process emphasizes operator precedence and the incremental expression solving performing internally by the computer. Students have the option of selecting any variable or operator in the expression. The tool requires that the "drill-down" proceed in the order it would appear in the abstract syntax tree. For example, in the expression "x / y", the student must select the division operator first, and only then may they select then enter the value x. If the student selects an incorrect sub-expression or node, they are guided to the correct ordering.

Correctly entered values, resulting from variables or sub-expression solutions, are substituted into the expression as the student enters them (See figure 3). At each step the student must enter a value or click on the next part of the program to run – this maintains student involvement and avoids "auto-pilot" where the student could click next or continue without thinking about what the system is doing.

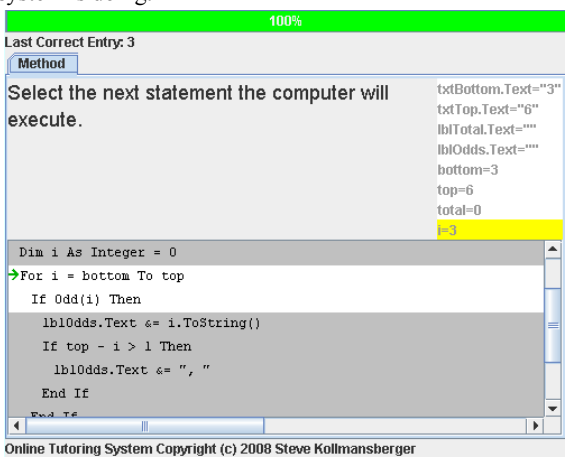

Figure 1. The student is prompted to click on the next step of a for loop.

If the student makes a mistake on selecting a portion of statement or expression, colored boxes (red and green) indicate the correct choice on mouse-over. If the student makes a mistake entering a value, the correct value is shown. The student still must take the action to click on the right entry or enter the given value. The errors decrease the score (shown in a colored bar along the top). Students may retry each exercise to increase their score, and indeed we found many students repeatedly did so.

The tool is written in Java and presented as an applet on a website which students access with an individual code. The tool ties into a web-based grading system and uses the student-provided access code to update their grade as they attempt various exercises.

The tool currently supports variables (integer, floating point, Boolean, and string), if and if-else statements, for and while loops (both pre and post test), and user defined functions. The tool does not yet support arrays, user defined classes/objects, or recursion.

2.1.Website

The exercises are available online, as is the tool, at http://www.kolls.net/ots. Visitors may select Anonymous access.

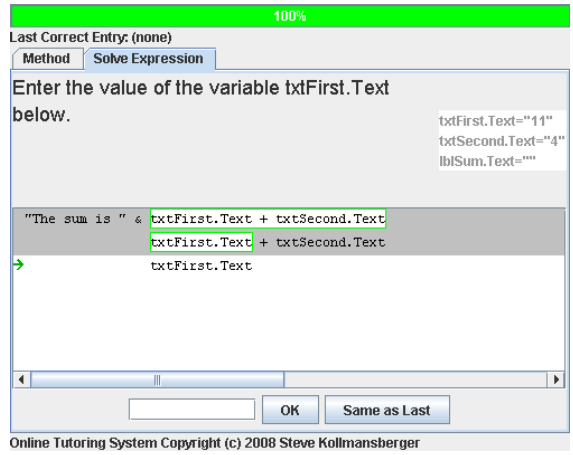

Figure 2. The student has selected expression subparts and is ready to enter a value. This expression is used to expose a common student misunderstanding: "+" does not always mean addition.

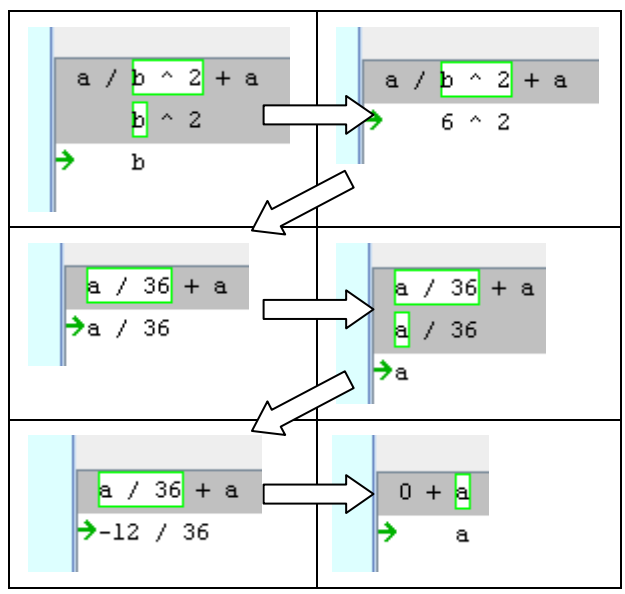

Figure 3. The process of solving an expression (given that b = 6 and a = -12, and both are integers). Some steps are omitted.

3. EVALUATION

We evaluated the tool in two sections of our introductory programming class as taught by two different instructors. In both cases, twelve exercises were used with due dates throughout the term, corresponding to the material being covered in class.

In order to establish the effectiveness of the tool, we compared the final grades and completion rate of those introductory sections which used the tool to those which did not. In addition, we

conducted two student surveys (one in each section that the tool was used) to ask the students their impressions of this technique.

3.1. Course Completion

The most significant change we found occurred in course completion. Course completion is measured by comparing the number of students who complete the first assignment with the number of students who complete the final exam. We exclude students who never attended or who never completed the first assignment, as these students would not have been exposed to the exercises and thus could not have been influenced by them.

Four sections (two from each instructor) which did not use the tool were measured; course completion in these sections ranged from 50% to 79%, with an aggregate of 66% of students completing the class. Two sections (one from each instructor) which did use the tool were measured; course completion in these sections was 89% and 95%, with an aggregate of 92% of students completing the class. Sections to use or not use the tool were selected arbitrarily, by the instructor, before each term began.

Use of the tool correlates with a 40% aggregate increase in class completion (See figure 4). However, all class sizes are fairly small (14-21 students), so statistical significance cannot be shown. One potential limiting factor includes the varied student make-up of each class. The results were gathered over the course of several terms, so the cohort position could also possibly have an effect.

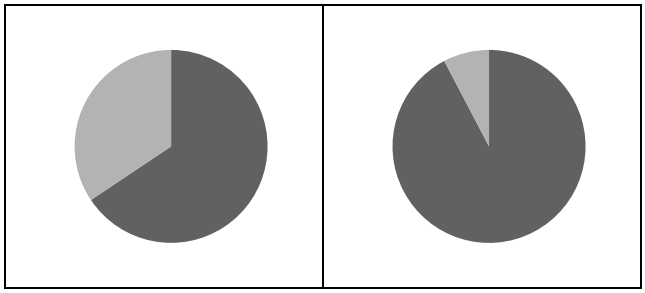

Figure 4. Course completion without tool (left) compared to course completion with tool (right). Darker area is proportion of students completing class.

3.2. Course Grades

In addition to being more likely to complete the course, we would expect that students with an improved mental model of computation would earn overall higher scores. We compared the final grades of students who completed the class for those sections which used the tool against those sections which did not.

Final average course grades for the four sections which did not use the tool were 71, 71, 74, and 80. Final average course grades for the two sections which did use the tool were 79 and 82. This improved represents an improvement of over 6 percentage points, on average, for sections which use the tool.

3.3. Student Survey

In one section, students were asked if they found the tool helpful, neutral, or harmful to their learning. Only 43% of students counted it helpful, with the most common complaint being that the exercises were "too repetitive."

In a separate, more detailed survey in another section, however, 73% of students said the exercises were "a good use of my time" and over half said they "helped me understand how a program executes." Moreover, 84% said the exercises should be continued in the introductory class and over three quarters said they should be extended and used in other classes as well. One student said the "step by step breakdown of how the computer executes the code" was very helpful. Students also indicated that they purposefully repeated the exercises to improve their conceptual understanding. These comments suggest that the tool does in fact help students build and improve a correct mental model of computation.

Students expressed a desire for better feedback from the system (why a choice was wrong, for instance), and also expressed concern over the length of some of the loop based exercises. Several other technical improvements, such as indicating all possible selections (currently a mouse-over is required) were suggested by students.

4. FUTURE WORK

Although the students may not always directly acknowledge the benefits of the exercises, objective course completion and grade data tends to support that the exercises do help students stay involved and learn in the class. However, these exercises as they exist now are only the tip of the iceberg for improving course completion and students mental models of computation.

Future work includes running additional sections with and without the exercises to see if the course completion difference is duplicated. A rigorous statistical analysis should be conducted, with a large enough sample size, to show if the improvement is significant. Student suggestions about the tool should also be taken into account; it should be extended to provide better feedback and explanations. In addition, the tool could track where students make errors and create a faculty report that allows the instructor to focus on "problem areas" in class. Other faculty agree this would be a dramatic improvement, allowing the results of the exercises to appropriately influence the use of class time.

Techniques for extending the tool to handle object oriented programming would allow it to be used in the more advanced classes. Students show significant confusion about OOP and having an interactive tracer may be very beneficial. Care needs to be taken to ensure helpful and straightforward representations of objects and classes. Work on representing object-oriented

program traces has been considered by Lange and Nakamura [7], and could possibly be integrated into an interactive tracing tool.

Finally, if the tool is to be used by larger audiences, accessibility concerns must be addressed. Currently, color distinction and mouse usage are both required. A modified version of the tool which supports purely keyboard input and provides an additional visual clue (besides red/green color) to incorrect/correct selections would be straight-forward to implement. Implementation of a tool which worked with assistive technology, such as screen readers, should also be possible.

5. CONCLUSION

We have demonstrated and evaluated a tool for helping students build mental models of computation. Our evaluation finds that student course completion in an introductory programming class increases dramatically when the tool is used. In addition, because the exercises are assigned as homework outside of class, and the results are automatically graded, the tool requires little investment of class time or instructor resources. Students may not "enjoy" using the tool, but they do agree it helps them understand program execution and build a mental model of computing.

We conclude by acknowledging that while the validity of an a priori consistent mental model as a predictor of student success is in doubt, the presence of a working model can only be helpful and indeed such a model can be developed by student effort through the use of an interactive tracing tool such as the one presented in this paper.

6. REFERENCES

[1] Boisvert, Charles. A Visualisation Tool for the Programming Process. ITiCSE, 2009.

[2] Bornat and Dehnadi. Mental Models, Consistency and Programming Aptitude. Australasian Computing Education, 2008.

[3] Bryant et al. Computational Thinking – What Is It, How Is It Relevant, Who's Doing What With It? Panel discussion at CCSC Northwest, 2009.

[4] Caspersen et al. Mental Models and Programming Aptitude. ACM SIGCSE, 2007.

[5] Dehnadi, S. Testing Programming Aptitude. Annual Workshop of the Psychology of Programming, 2006.

[6] Jin, Wei. Pre-programming Analysis Tutors Help Students Learn Basic Programming Concepts. ACM SIGCSE, 2008.

[7] Lange and Nakamura. Object-oriented Program Tracing and Visualization. Computer, May 1997.

[8] Reges, Stuart. The Mystery of "b := (b = false)". ACM SIGCSE, 2008.

[9] Terada, Minoru. ETV: A Program Trace Player for Students. ACM SIGCSE, 2005.

[10] Vainio and Sajaniemi. Factors in Novice Programmers' Poor Tracing Skills. ACM SIGCSE, 2007.