ABSTRACT
Often when designing an educational tool, the focus is primarily on how well the tool helps the student learn a concept. However, always in educational research there is an underlying desire to determine what factors actually influence student learning. This is because an understanding of these factors can lead to the design of more effective tools/techniques. The focus of our research has been on developing a tool to help students learn algorithm design. The ability to design an algorithm for a given problem is one of the most important, and unfortunately one of the most difficult to accomplish, learning outcomes of computer science courses. It has previously been shown [13] that students who use AlgoTutor, a Web-based algorithm development tutor, are significantly more likely to think that algorithm design prior to coding is important and to have confidence in their own ability to design an algorithm. From follow up studies, we have found that students who have used AlgoTutor in introductory computer science classes are not only more confident in their ability to design an algorithm, but also more likely to design a correct algorithm than those who have not used AlgoTutor. Additionally, we show that the course management utility for the AlgoTutor system can be used to investigate questions about factors that influence student learning. As an example we investigate the question, “how much is too much help and how much is not enough help if a student is having difficulty solving a problem?”

Categories and Subject Descriptors

General Terms
Design, Management, Human Factors.

Keywords
algorithm development, learning environment, Web-based tutoring system.
1. INTRODUCTION
As educators, our goal is to convey concepts to students in such a way that students can understand and apply the concepts. Because of this goal, we are continuously evaluating two things: 1) how well the students grasp the concepts; and 2) what is the best way to present/teach the concepts. While the evaluation of the students' understanding for a specific concept is tied to the effectiveness of the learning for that concept, the best practices for presentation may actually be applicable to many concepts. The former evaluation is definitely important, but the latter may, in the long run, be the more valuable. If teachers understand how students learn, it could certainly help teachers present the concepts effectively. In this paper we discuss both how AlgoTutor [8, 11, 12] can help students learn to design algorithms and how it can also help teachers understand how students learn.

1.1. Motivations/Problems
It has been known that teaching problem solving techniques is not an easy task [1, 2, 4, 6]. Additionally, anecdotal evidence tells us that students in introductory computer science classes feel that there is no need of an algorithm because the problems they solve are relatively simple. This attitude prevents them from building a habit of developing algorithms prior to coding and from acquiring algorithm development skills. There has been a lot of interest in engaging students in programming activities by using visual systems such as Alice or Scratch [3, 5, 9] and addressing the importance of algorithm development [7, 10]. However, prior to the development of AlgoTutor [8, 11], there was no tool for practicing algorithm development as a prerequisite of coding in addition to helping them learn algorithm development skills.

Even though, we stress the importance of the algorithm development in lectures, we found out students often don’t know how to develop an algorithm. If they are required to submit an algorithm for a lab assignment, they will turn in either a completed program or a nearly meaningless outline. This indicates that students often do not have an understanding of the basic concepts of algorithm design [8].

For example, when we asked our students to write an algorithm, one of four things usually happens.

1. Some students just turn in the program.
2. For those who actually try to write an algorithm, some just repeat the problem description,
3. some are completely wrong, and
4. some would be correct if the right variables were used.

Table 1 shows three representative incorrect student algorithms. The algorithm in Sample 1 is a mere restatement of the problem itself. The algorithm in Sample 2 is completely incorrect, showing the student could not choose the correct operations including missing the iteration step necessary to search the array. The algorithm in Sample 3 shows the student basically knew which operations to use, but could not appropriately map the parameters.

<table>
<thead>
<tr>
<th>Sample 1: Sort an array of N numbers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use a bubble or selection sort to sort the array from lowest to highest</td>
</tr>
<tr>
<td>2. return and print the array to the screen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample 2: Find the smallest value in an array:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. if first &lt; second</td>
</tr>
<tr>
<td>lowest is the smallest of first and third</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>lowest is the largest of second and third</td>
</tr>
<tr>
<td>2. return the lowest value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample 3: Find an average of 10 numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. while value &lt; 10</td>
</tr>
<tr>
<td>value1 + value2</td>
</tr>
<tr>
<td>value ++</td>
</tr>
<tr>
<td>value total / 10 = Average Value</td>
</tr>
<tr>
<td>2. return Average Value</td>
</tr>
</tbody>
</table>

Table 1. Samples of incorrect student algorithms.
1.2. Our Solutions
We developed the AlgoTutor system to address problems described in the previous section. AlgoTutor is a Web-based tutoring system that enables introductory computer-programming students to practice algorithm development using the top-down design strategy with abstraction refinement. The system also allows students to practice the bottom-up approach by reusing the algorithms created previously. The system provides online grading and feedback on mistakes. AlgoTutor also addresses other issues that arise in algorithm development (e.g., verifying the correctness of an algorithm or converting an algorithm to code). Since AlgoTutor is a Web-based system, it allows easy deployment and access from locations outside the lab. Because all data is stored centrally, it is easy to develop tools for instructors to analyze student performance and behavior.

We used the AlgoTutor system in CS-I and II since 2009, and we found out that students who use AlgoTutor are significantly more likely to think that algorithm design prior to coding is important and to have confidence in their own ability to design an algorithm [13]. In the paper [13] we show that this change in attitude is actually an indicator of a change in ability. Not only are students who have used AlgoTutor more confident in their ability to design an algorithm, but also they are more likely to design a correct algorithm than those who have not used AlgoTutor.

Figure 1 shows a typical learning cycle. Once a material is presented in a lecture, an assignment or homework is given to students. This process will help students understand what they need to know and realize what they don’t understand. AlgoTutor provides an immediate feedback to help student identify their problems and guides students by showing structured algorithm development process.

Additionally, AlgoTutor also contains the Course Management module which is specifically for teachers. This module allows teachers to set up accounts, design problems, review grades, and visualize student behavior. The visualization tool has brought to light some interesting patterns in student behavior. In this paper we will discuss one such example that is related to the question, how much help do you give a student who is having difficulty solving a problem.

In the following section, we describe the system architecture of AlgoTutor. Section 3 describes the various components of AlgoTutor and how they relate to the steps involved in problem solving using a computer programming language. In Section 4, we report on the improvement in student performance caused by using AlgoTutor. Section 5 discusses the utilization of the course management tool to evaluate what factors influence student learning. We conclude with future directions of the research.

2. SYSTEM ARCHITECTURE
The Algoututor system is a client-server architecture made up of several modules as illustrated in Figure 2. The client side provides the student and the instructor interfaces. The instructor interface allows the teacher to create various problems along with their associated solutions and they are stored in the system database. The student interface allows the student to use different tutor components which access the database and presents these problems to the student so that they can develop an algorithm for the problems.
For students’ algorithm development process, the system provides them with various tools including Algorithm composer, Algorithm tracer, and ProgramPad. Algorithm composer allows students to construct their algorithms by selecting an appropriate set of operations for a given problem from a list of instructor-provided-operations and ordering them. The algorithm tracer allows students to verify the correctness of their algorithm by providing visual tracing of the algorithm developed by the students. ProgramPad guides the student to convert the constructed algorithm to an actual C++ code.

Through the instructor interface the teacher accesses the course management module, which allows the teacher to set up user accounts and create labs. To create a lab the teacher creates various problems and adds them to the lab. When making a problem the teacher also prepares a set of solutions (or algorithms) each of which is in a form of a sequence of operations. In addition, the course management system provides a visualization tool where the teacher can monitor the student activities.

All the software used in AlgoTutor, such as MySQL, Apache, PHP, is 100% open source software. The AlgoTutor production server was built on the Linux operating system. However, using WampServer or XAMPP, one can reproduce the production server on the Windows and Mac operating system. The client interface is a web-based graphical user interface that supports drag-and-drop operations. It was developed using the PHP scripting language to generate XHTML web pages. Cascading Style Sheet (CSS) was used to format the web page colors, fonts, layout and other aspects of web page presentation. JavaScript was used to enable drag-and-drop user interaction and asynchronous HTTP requests via Ajax. The Ajax programming gives a lot of power and flexibility for developing browser-based user-interfaces that provide user experiences that closely resemble desktop applications. The use of the Ajax technology provides an interactive and responsive browser user interface. The student access to the Algotutor system is controlled by the Apache and PHP authentication system and each student is identified by using a user id and a password. To limit and control the access to other user’s solutions and the course management software, three levels of access rights have been created: instructor, assistant, and student access. The user identification is maintained by using cookies during each session.

3. TUTORS
3.1. Algorithm Composer
In this subsection, we introduce the functionality of Algorithm Composer using a sample Algotutor session.

When a student logs onto AlgoTutor to design an algorithm for a particular problem, the first screen they see describes the top-level problem to be solved along with a link to the page that will allow them to begin developing
the algorithm. Figure 3 shows a sample initial screen. The problem description is in the left column of the table while the link to start algorithm development is the word solve in the right hand column.

When the solve link is clicked, the student will be directed to a page similar to the one shown in Figure 4. The student is now in the operation selection phase of the algorithm. The problem description is at the top of the screen for reference. Along the right hand side of the screen is a list of the operations that will be used in designing the algorithm for this particular problem. One difference between the instructor screen and the student screen is that the instructor has access to all operations in the database whereas the student only has access to those that pertain to the given problem. The operations are divided into top-level operations and lower level operations. The lower level operations are ones that are used inside of a top-level operation. For example, add next value to sum would be part of a loop that is defined by while counter is less than n add it to sum.

**Algorithm Tutor**
Welcome, Chrisia Pence. Logout Feedback

**Problems**

<table>
<thead>
<tr>
<th>description</th>
<th>operations correct</th>
<th>parameters correct</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level 1:</strong> Assumptions for integer values m and n have been read (given).</td>
<td>not solved</td>
<td>not solved</td>
</tr>
<tr>
<td>1. Uses the while loop to find the sum of integers m through n inclusive.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Displays the resulting sum.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Examples:</strong> Summation of integers 1 to 5 is 15 (1+2+3+4+5).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summation of integers 1 to 10 is 55.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summation of integers 10 to 13 is 46.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3:** The initial problem presentation screen.

**Problem Description**
Assumptions for integer values m and n have been read (given).
1. Uses the while loop to find the sum of integers m through n inclusive.
2. Displays the resulting sum.

**Operations**
- top level operations
  - Initialize counter
  - Initialize sum
  - print the sum
  - while counter is less than n add it to sum
- lower level operations (used below top level)
  - ( ) < ( ) (less than or equal to)
  - add next value to sum
  - increment counter by 1

**Variables**

**Figure 4:** The initial operation selection phase screen. Operations which should be used to solve the given problem are shown on the right.
In order to develop an algorithm, the user drags the operations from the right side to the left side. The selected operations that are on the left can be reordered by drag-and-drop. There are buttons at the bottom of the screen that enable the student to check the correctness of the current ordering of the selected operations, read help pages, or send questions to the instructor. Clicking the Check my ordering button will provide immediate feedback (See Figure 5.) Figure 6 shows the operation selection phase after the student has dragged all operations to the left. Notice that the first two operations, initialize counter and initialize sum, actually have the same underlying operation assign a value to a variable. The underlying operation is the only one in the AlgoTutor database, but an instructor is allowed to add comments to an operation to make it problem specific, and those comments are what the student will see.

Figure 5: The student solution is graded each time the Check my ordering or Check my mapping button is pressed.

Figure 6: The operation selection phase screen after dragging operations from right to left to create an algorithm that solves the given problem.

Also note in Figure 6 is the presence of several question marks in the algorithm. These question marks represent the variables in the generic operation that need to be mapped to problem specific values. When a student confirms the algorithm is correct (usually by receiving a grade of 100% after clicking the Check my ordering button) or the student wants to move on to the next phase, then they can click the Ordering done, parameter mapping next button. At this point, the student’s solution is checked against the set of the instructor’s solutions. An instructor solution may have a par-block (parallel execution block) to group operations which are not order dependent. This block is not shown to students in the operation selection phase, and this feature allows a single instructor solution to cover
many alternative correct solutions. Now, the grade of the student’s algorithm is recorded in the database, the standard solution, which is one of the instructor’s solutions, is presented for the parameter mapping phase.

**Figure 7**: The parameter mapping phase. Students replace question marks with problem specific values.

<table>
<thead>
<tr>
<th>Algorithm Tutor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Welcome, Chrisia Petey. Logout Feedback</strong></td>
</tr>
<tr>
<td><strong>Problems</strong></td>
</tr>
<tr>
<td>Progress: [ ] [ ] [ ] [ ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>level 1:</th>
<th>description</th>
<th>operations correct</th>
<th>parameters correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is part of OLA6. This function will be called for evaluating one arithmetic expression written in infix notation.</td>
<td>Evaluate an infix algebraic expression by converting the infix expression to postfix expression and then evaluate the postfix expression using stack. At the end of your algorithm, print the index expression, postfix expression, and the final result of the algebraic expression in the order. Check assignment description for the assumptions on the infix algebraic expression.</td>
<td>100 % correct</td>
<td>100 % correct</td>
</tr>
<tr>
<td>Level 1:</td>
<td>Convert infix string to postfix. You should structure your cases as following order:</td>
<td>not solved</td>
<td>not solved</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>level 1:</th>
<th>description</th>
<th>operations correct</th>
<th>parameters correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1:</td>
<td>Evaluate postfix string and return result of evaluation</td>
<td>not solved</td>
<td>not solved</td>
</tr>
</tbody>
</table>
In the parameter mapping phase, students replace the question marks with problem specific values. Figure 7 shows an initial screen of a parameter mapping phase. Students can replace each question mark either by clicking on the question mark and selecting an appropriate value from a pop-up menu or by dragging a variable tile from the right side to the question mark. Vertical bars around variables or literals indicate that the instructor locked those variables so they are not editable by the student. This eliminates the need for some alternative solutions and helps students figure out appropriate parameter mappings. For example, in Figure 7 it is correct to initialize x before you initialize sum, but the locking forces the student to do it in the order given while at the same time providing some hints for a correct mapping.

After the student finishes the parameter mapping phase, AlgoTutor will display the web page illustrated in Figure 3, but the progress bar would be filled in, the grades for each phase would be displayed, and the solve link would be replaced by View Solution Done. The View Solution link enables the students to view the correct solution. While the example in this section was a simple example, AlgoTutor does support complex problems containing multiple levels of sub-problems. For a complex problem, the development steps for abstract level 1 would be like those shown in Figure 3 – Figure 7. After finishing the parameter mapping phase of level 1, AlgoTutor would display a screen like that is shown in Figure 8 and students can then follow the links to develop the algorithms for the sub-problems. Solving a sub-problem prior to solving its parent problem is a form of the bottom-up approach because the solution to the parent problem reuses the previous solutions of the sub-problems.

### 3.2. Visual Algorithm Tracer

We have developed an algorithm tracer that allows students to verify the correctness of their algorithm or to locate possible errors by providing visual tracing of the student algorithm. The visual algorithm tracer helps students gain a better understanding of algorithm execution. To run the algorithm tracer, students either select test data from a pool of data sets or provide their own test data for each variable used in the algorithm. The algorithm tracer executes the algorithm by stepping through an operation at a time so that the student can follow the control flow and monitor changes in variable values.

Figure 9 shows a snapshot of the algorithm tracer which steps through an algorithm for the problem of “reading numbers until zero is read using a loop.” We observe that the algorithm tracer interface is divided into four panes: the algorithm pane, the variable pane, the output pane and the control pane. The algorithm pane shows a listing of the current algorithm in a format similar to that of the composer. Operations are numbered for reference during execution of the algorithm. The next operation to be executed is highlighted as shown in Figure 9 (step 3). The variable pane shows a listing of all currently defined local variables with their current values. The last operation step that modified the value of that variable is shown in parenthesis to the right of the value. (For input problem parameters if they exist, “input “ is shown instead.) The output pane shows output from the execution as it progresses.

Students may submit their algorithms for grading by clicking the “Submit for grading” button. The student’s algorithm is then run against a known correct solution for all predefined test cases. The outputs for each test case are compared and the student receives a score that is the percentage of all test cases that matched. These grading runs are performed without visualization, i.e. no diagnostic information will be provided. The student may submit any number of times, with only the last submission used for final scoring. Students can also execute the algorithm with selected data to see the output of the execution without tracing by clicking the “Execute w/ selected” button. Note that the algorithm tracer can only be used for a certain category of problems whose algorithms consist of traceable operations. Traceable operations are those operations that are implemented so that the results of each operation can be visualized with the actual input values for each variable.

**Figure 8:** When the top-level design is complete, students are presented with the next level.
3.3. ProgramPad

The algorithm developed using the composer eventually needs to be converted to program code. Students in CS-1 classes often do not see the relationship between the algorithm and the code. For this reason, we develop the ProgramPad component. ProgramPad converts most of the operations/steps in the algorithm into C++ code, and provides an interface for completing the program.

Figure 10 shows a snapshot of ProgramPad for an algorithm. ProgramPad displays the problem description in the top pane. The problem parameters and local variables used are displayed in the variable pane on the right. The program pane on the left shows the C++ code for the student algorithm developed in the composer. ProgramPad provides capabilities for edit, save, load, build, run, and print.

Each operation in the Composer has corresponding C++ code. For example, “print variable” corresponds to “cout << variable;” and “print a right triangle of height h with symbol s” to “void PrintTriangle (int h, char s).” Thus we can convert an algorithm with abstract operations into a program with multiple functions.

ProgramPad can also be used to address more advanced programming concepts such as parameter passing between functions (differentiating between the function parameters and local variables and/or between arguments in a function call and parameters in the corresponding function definition). In addition, ProgramPad can be used to teach testing functions using a driver and functional decomposition.

![Figure 9: A snapshot of algorithm tracer](image)

![Figure 10. A snapshot of ProgramPad for the algorithm in Figure 3.](image)
4. IMPROVEMENT IN STUDENT PERFORMANCE

In the paper [13], we reported that AlgoTutor can change students’ attitudes toward algorithm development; students who used AlgoTutor in CS-1 were more likely to realize the importance of algorithm design in problem solving and to have confidence in their own algorithm development abilities.

This section discusses the experiments conducted to evaluate how well AlgoTutor help student performance.

4.1. Experiments in the CS-1 Course

This experiment was conducted for three CS-1 classes taught by the same instructor in Fall 2011. We selected one class as the experimental group and the other two classes as the control group. The experimental group used the AlgoTutor system, and the control group did not. All the other teaching material and learning environments were the same; the students in both groups were given the same lectures, homework, exams, and programming assignments. The same programming assignments were given to both groups with approximately two weeks allowed for completion of each assignment. The students were to submit the algorithm along with the code in text files for each of the programming assignments. The students in both groups received feedback on the evaluation of their algorithms and code using the same rubric. To measure the effectiveness of the AlgoTutor system, we have students in the experimental group use the system before they wrote code for the first seven programming assignments and not use them for the last one. The exercises in the AlgoTutor lab let the students compose an algorithm for the main problem and/or sub-problems for the programming assignments. For the last programming assignment, however, the students in the experimental group were not provided AlgoTutor exercises; they designed their algorithms on their own in the same way as the students in the control group. Table 2 shows the comparison of the average exam scores, algorithm design scores, and program scores between the two groups.

<table>
<thead>
<tr>
<th></th>
<th>Exam</th>
<th>Algorithm</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgoTutor group</td>
<td>77.5</td>
<td>81.0</td>
<td>91.8</td>
</tr>
<tr>
<td>Non-AlgoTutor group</td>
<td>86.8</td>
<td>71.5</td>
<td>80.2</td>
</tr>
</tbody>
</table>

Table 2. The comparison of student performance.

During the semester, we found the pre-existing performance difference between the two groups. We observed that the average score of the first exam for the control group was higher than the experimental group; this was before the use of the algorithm tutor. We observe that the algorithm design score and the program score averages of the AlgoTutor group are higher than those of the Non-Algotutor group. The difference in the mean scores for the algorithm design is 9.5% and is highly significant by a t-Test for the two groups with 95% confidence level. The data analysis using a t-Test conducted for the program scores shows that they are significantly different as well with 99% confidence level. This is particularly interesting in light of the fact that the exam scores for the AlgoTutor group were generally lower than those for the control group. From this analysis, we believe that the AlgoTutor system has a positive effect on the student’s performance on algorithm design and implementation, and this positive effect is apparently not linked to how well a student can memorize facts or take tests.

4.2. Experiments in the Advanced Data Structures Course

During the last several years, several CS-1 and CS-2 instructors have incorporated AlgoTutor into their lectures, labs and programming projects. To evaluate the effectiveness of AlgoTutor in helping students learn algorithm development skills, an experiment was conducted in the Advanced Data Structures course. During the first week of Fall 2011, a questionnaire was given to students in two sections. Besides collecting the data about students’ experience with AlgoTutor, the questionnaire also asked students to design an algorithm for the following problem:

*Given two sets $A$ and $B$, the intersection of $A$ and $B$, denoted by $A \cap B$, is a set having those members which are in both sets. For example, given $A = \{2, 3, 5\}$ and $B = \{7, 5, 2\}$, $A \cap B = \{5, 2\}$. Please write an algorithm in pseudo-code to find the set intersection $A \cap B$ for sets $A$ and $B$. Start with a top-level algorithm and apply step-wise refinement.*

A sample step-wise refinement for a simple problem was given as a guideline. We expect that students taking this course should know the solution to the set intersection problem. The purpose of the question was to evaluate how
well they can use pseudo-code to formulate and refine the algorithm. An instructor graded all algorithms using the same rubric.

All participating students were divided into two groups: Group 1 consisted of students who had been exposed to AlgoTutor in CS-1 and/or CS-2, and Group 2 consisted of students who had never used AlgoTutor. Table 3 shows the comparison of average scores for top-level and refined level algorithms between the two groups.

These two groups are comparable based on the number of students in each group and their grades in CS-1 and CS-2. The average grade for Group 1 is 3.66 for CS-1 and 3.25 for CS-2; and 3.62 for CS-1 and 3.26 for CS-2 for Group 2. However, there is a significant performance difference on the algorithm design question between these two groups. Group 1 had much higher grades on both levels: the top level algorithm design and its refinement, although only the difference on refined level grades(*) is statistically significant using a t-test with 95% confidence level.

This experiment clearly supports our conjecture: teaching algorithm design early in the CS curriculum does help students learn algorithm development skills, and AlgoTutor is an appropriate tool for algorithm design training.

<table>
<thead>
<tr>
<th></th>
<th>Group 1 (AlgoTutor)</th>
<th>Group 2 (No AlgoTutor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-level Grade</td>
<td>91%</td>
<td>77%</td>
</tr>
<tr>
<td>Refined level Grade (*)</td>
<td>76%</td>
<td>55%</td>
</tr>
<tr>
<td>CS-1 Grade average</td>
<td>3.66</td>
<td>3.62</td>
</tr>
<tr>
<td>CS-2 Grade average</td>
<td>3.25</td>
<td>3.26</td>
</tr>
<tr>
<td># students</td>
<td>20</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 3. Performance of Students in advanced data structures.

5. EVALUATION OF LEARNING FACTORS

In addition to the modules for students to use in algorithm development, the AlgoTutor system contains several modules for course management. The Course Management modules have been developed specifically for teachers. The system allows teachers to set up and manage accounts, design problems and labs, review grades, and visualize student behavior. The visualization tool has brought to light some interesting patterns in student behavior. In this section, we will discuss one such example that is related to the question, how much help should we give a student who is having difficulty solving a problem?

The AlgoTutor system allows students to check the correctness of their algorithms. If a student clicks the “check my ordering” button, AlgoTutor points out the first misplaced operation if one exists. In the early system development stage of AlgoTutor, there were no limits on how many times a student could check the correctness of their algorithm. Consequently, some students took advantage of this and checked the correctness of their algorithm an unreasonable number of times instead of actively thinking about the problem. For example, given a problem whose solution algorithm only contained 10 steps, one student clicked the “check my ordering” button 188 times. Obviously, that student did not actively think to solve the problem; instead he/she tried different combinations of suggested steps for this problem. We noticed this excessive number of checks while examining student activities using our visualization tool. Figure 11 shows the snapshot of a few students’ activities for this problem using the visualization tool. Each data point in a graph represents a student’s activity while developing an algorithm. For example, the red line in the graph shows scores as students click while the bar graph shows the interval between clicks. Click numbers are shown in the x-coordinate.
This finding led us to impose a check limit for each problem in order to promote active thinking. However, this caused two new issues: 1) the problem of determining an optimal number for the check limit; and 2) mitigating student frustration caused by the check limit. To alleviate student frustration, we introduced a lifeline in the system. When a student gets stuck in algorithm development, the student can use a lifeline which will provide the correct next step in the algorithm. The lifeline will help the student proceed to the next step thereby avoiding too much frustration. A teacher can set the number of lifelines for each problem as well as setting penalty points to avoid excessive use of the lifelines.

We believe there is no magic number for the check limit that will fit all problems. The proper check limit should depend on the complexity of the problem. To reveal the relationship between the check limit and the complexity of the problem, an experiment was conducted in Fall 2011. In the experiment, students were randomly assigned to one of three groups. All the students in a group had the same check limit that was one of 3 values: 50%, 100%, or 200% of the number of steps in the solution algorithm. If one group got 50% for the first problem, they got 100% for the second problem, and 200% for the next. This round-robin process repeated during the semester. We used this scheme in Fall 2011 and analyzed the student activities and performances using the Student Activity Monitoring module.

We found that the students who got a higher check limit tended to spend a longer time and check the correctness more frequently. However, they ended up getting lower scores. We believe that too much help (i.e., a higher check limit) caused students to rely on the automatic feedback instead of promoting active thinking. We are planning to set the check limit to be 75% of the number of steps in the solution algorithm for each problem in Spring 2012. We are planning on continuously monitoring student activities and their performance for future adjustments.

6. CONCLUSION AND FUTURE WORK
The ability to design an algorithm for a problem is one of the most important, and unfortunately one of the most difficult to accomplish, learning outcomes of computer science courses. Even when students can be convinced that algorithm design is important, they frequently are not confident that they can design an algorithm. It has previously been shown [13] that students who use AlgoTutor, a Web-based algorithm development tutor, are significantly more likely to think that algorithm design prior to coding is important and to have confidence in their own ability to design an algorithm.
In this paper we have shown that this change in attitude is actually an indicator of a change in ability. Not only are students who have used AlgoTutor more confident in their ability to design an algorithm, they are more likely to design a correct algorithm than those who have not used AlgoTutor. Additionally, we show that the AlgoTutor Course Management Utility can be used to investigate questions about factors that influence student learning. We have analyzed student activities and observed the system parameters such as how the check limit affects student behavior (e.g., their performance and the time spent on exercises).

Based on our findings, we are planning to set the check limit to be relatively small so that students will spend less time clicking on the check button and more time actively thinking about how to solve the problem. We are planning to continuously monitor student activities and make future adjustments accordingly. We also want to add a time limit for each problem while proposing optional reinforcement sessions for the students who need more help. Another possibility is to give students an option to adjust the time limit on their own. Whatever changes we make in the future, we are confident that AlgoTutor can teach students algorithm development skills.

7. ACKNOWLEDGMENTS
This research was supported by the NSF CCLI grant DUE-0737257.

8. REFERENCES