Effectiveness of Cognitive Apprenticeship Learning (CAL) and Cognitive Tutors (CT) for Problem Solving Using Fundamental Programming Concepts

Wei Jin
Dept. of Computer Information Sciences
Shaw University
Raleigh, NC 27601
(919) 546-8376
wjin@shawu.edu

Albert Corbett
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-8808
corbett@cmu.edu

ABSTRACT
In this paper, we describe our approach in addressing learning challenges students experience in introductory programming courses. We combine two effective instructional methodologies to help students learn to plan programs prior to writing code: Cognitive Apprenticeship Learning (CAL) and Cognitive Tutors (CT). In the CAL component, the instructor models program planning in class and paper handouts are used to scaffold program planning in homework assignments. In CAL-CT, the program-planning process is also supported by a computer tutor which provides step-by-step feedback and advice. The results show that the combined CAL-CT approach yielded substantial gains over traditional instruction. Its advantage over the CAL-Only approach is also significant.

Categories and Subject Descriptors
K.3.2 [**Computers & Education**]: Computer & Information Science Education – *Computer Science Education, Curriculum.*

General Terms
Design, Experimentation, Human Factors

Keywords
CS1, Cognitive Apprenticeship Learning, Cognitive Tutors, Introduction to Programming, Pedagogy, Scaffolding

1. INTRODUCTION
Shaw University is the oldest Historically Black University in the south and its mission is to provide higher education to those who otherwise would not have such an opportunity. Since there are substantial performance gaps among students and many students are not well-prepared for college, it faces special challenges in helping all students to learn and progress.

Our project started with the observation that many of our students specifically "freeze" and do not know where to start with programming assignments. Many educators have observed the difficulties that students have with mastering programming [14] and many innovative approaches have been proposed. We had previously tried several methods, such as peer collaboration, which may be effective for different student population, but we did not observe much effectiveness.

Our approach is to use computer tutors to support students and help them develop problem solving strategies that they can use independently in the future. These tutors simulate a human instructor who demonstrates how to approach a programming problem and guides students through problem solving processes. Through numerous guided practices, we hope that eventually the problem analysis and solving process will become students' habits and they can use them automatically in their independent problem solving.

This approach fits our students' needs. Guidance tutors, which start students off with a problem analysis, then guide students through an expert problem-solving process in a step-by-step fashion, seem to be the most appropriate. As it turns out, guidance tutors are relatively easy to implement, because they follow specific problem solving strategies.

In fact, our approach is a combination of the Cognitive Apprenticeship Learning (CAL) model and Cognitive Tutors (CT). Both have been shown effective at helping students improve problem solving skills in domains that require abstract thinking (see Section 2 for detail).

The study presented in this paper is an extension of previous work [11], which focused on the very basic programming concepts – variables and basic statements. First, in addition to variables and basic statements, currently we have developed computer tutors for if statements and for loops. Second, the tutors in the previous report [11] only provide students with planning guidelines on how to solve a problem (writing a program). In contrast, our current tutors will help students step-by-step in planning and writing complete programs.

In this paper we report an evaluation of three learning approaches, CAL-CT (CAL with Cognitive Tutor support), CAL-Only (CAL with paper worksheets) and no-CAL (conventional problem solving), based on student test scores collected over several semesters. The data show that both the CAL-CT and the CAL approach have improved student test scores. On average, test scores for students in the CAL-CT group are 53% higher than

SIGCSE'11, March 9–12, 2011, Dallas, Texas, USA.

students that benefitted from CAL-Only instruction and 64% higher than students in the no-CAL group.

The remainder of the paper is organized as follows: In Section 2, we discuss the rationales for our approach in the context of CAL and CT and other related work. We describe the CAL component in Section 3 and the CT component in Section 4. We evaluate the effectiveness of CAL and CAL-CT in Section 5. We will summarize conclusions and future work in Section 6.

2. RATIONALES AND RELATED WORK

2.1 Cognitive Apprenticeship Learning (CAL)

Cognitive apprenticeship learning (CAL) models traditional apprenticeship learning, which has successfully trained individuals in various trades and professions [5]. Students learn complex skills by practicing in real life problem-solving settings. In addition to explaining how to perform a task, the teacher demonstrates how to perform the task (modeling), gives individualized guidance and feedback to the apprentice during practice (coaching), and gradually reduces the amount of help given as the apprentice develops proficiency in the task (fading). During this process, the teacher also provides scaffolding tools to guide the apprentice in the performance of the task, which are discarded as the apprentice becomes more adept.

The cognitive apprenticeship learning model helps students acquire abstract skills. Since abstract skills engage thinking processes that are "invisible," the teacher articulates expert thought processes to students. The description of these problem-solving processes and strategies provides organizational scaffolds for students and helps them acquire higher-level skills. In addition, instructors may use other materials, such as specially designed problems or worksheets to guide the student during practice [9]. In contrast, traditional forms of instruction are much less explicit with the problem-solving process. They often present students with models of complete answers that give no indication of the decision-making process and have little or no coaching component.

2.2 Cognitive Tutors (CT)

Cognitive tutors are an educational technology with characteristics similar to human tutors. They are most effective at developing skills that can only be gained through applying knowledge in a problem solving context. These skills are often the higher-level skills in Bloom's Taxonomy [3], such as knowledge application and synthesis, which are the skill levels that students need to reach to master programming.

Cognitive tutors pose complex problem-solving tasks to students and provide the individualized advice students need to succeed [1, 2, 6, 12]. Cognitive tutors are based on an understanding of both domain knowledge and problem-solving strategies. This knowledge is represented as a "cognitive model" in the cognitive tutor [1], an expert system which can solve the problems in the many ways that students solve them. As a student works, the cognitive model is used to follow the student's step-by-step solution and provide accuracy feedback. The tutor does not automatically provide detailed advice; instead students have the opportunity to reflect on and correct their own mistakes. However, if the student asks, the cognitive model is also employed to provide advice on how to accomplish each problem-solving step.

Cognitive tutors have been shown to speed learning by as much as a factor of three and yield an achievement effect size of about one-standard deviation compared to conventional instruction [2]. This is about twice the effect of typical human tutors [7] and about half the effect of the best human tutors [4].

2.3 Our Approach: CAL-CT

The cognitive apprenticeship learning model is an instruction methodology; cognitive tutors are computer programs. Combining these two approaches has advantages over using either approach alone. The advantages of CAL-CT over CAL-Only are as follows:

- Cognitive tutors make it possible for students to receive more individualized "coaching" outside of classrooms/office hours.
- Use of automatic tutors encourages students to actively participate in learning by limiting the fear of mistakes.
- Availability of help is immediate, which is made possible by the hints made available to students by automatic tutors.

2.4 Other Related Work

There have been various related approaches to automatic tutoring in programming that have had some success, for example, natural language dialog to help students develop pseudo code [13], reverse engineering to determine whether a program satisfies goals and then provide relevant hints [18], use of filling-blanks to focus on crucial part of the program [15], debugging tutors to help master programming concepts [8], and machine learning to determine grasp of knowledge and present learning materials adaptively [17]. Some enhanced IDEs also have tutoring features, such as identifying syntax/logic/style errors and providing feedback that is more meaningful [10, 16].

3. THE CAL COMPONENT

The CAL component is a step-by-step problem solving process to guide students in the right direction and help them develop habits to think as experienced programmers. We have developed instruments (e.g. worksheets) that help students follow this process. We emphasize that these forms are effective only if instructors model how to use them in problem solving through multiple examples and demonstrate how to convert information in a worksheet into a program.

3.1 Variables and Basic Statements

Problem solving involves three steps: (1) mental visualization of how the program interacts with a user, (2) variable analysis, and (3) flow analysis. Step 2 extracts essential information from step 1: the inputs the user will provide for the program and the outputs the program will calculate and give to the user. Step 3 determines the order of data processing. We previously developed a worksheet that explicitly supports variable analysis and flow analysis activities in program design, as described in [11]. For variables and basic statements, a paper-and-pencil worksheet was developed with three sections: Program I/O, Variables, and Flow. In the first section, students determine the relevant data items and whether each data item is an input or output of the program. In the next section, students choose a variable name and type for each data item. In the third section, students determine the program actions, which include getting input values from user and computing and displaying output values. The worksheet provides the framework for students to think. It is most effective to introduce the worksheets in the classroom where an instructor

models their use and summarizes after students spend some time working on a sheet.

3.2 If Statements

Figure 1 shows a correctly completed CAL worksheet for an IF statement. The problem description and the shaded areas are provided to students and the rest are blanks that students need to fill out. This worksheet again divides the program planning process into several sections: (1) Case analysis: A case is the set of situations under which the program should perform the same actions. Eventually a case will be specified by a logic expression, but during step 1, students describe each case in plain English. (2) Exclusiveness/order analysis: Students indicate if the cases in the program are exclusive of each other (that is, if only one can be true at any given situation). Exclusive cases can be implemented as one multi-branch if statement. If cases are non-exclusive, they should be implemented as independent if statements. (3) Logic expressions and actions: Students convert their English case descriptions into formal symbolic expressions for the tests and actions and at the bottom of the page assemble these components into one or more if statements.

Since variables are a topic for Basic Statements, when students reach the topic of if statements, they already achieved certain fluency. Here students have freedom to choose a proper variable name for a data item as long as consistency is maintained throughout the worksheet. Students also need to bring their work product (in the last blank space) into a program to compile, debug, and test. The worksheet itself only provides a framework for students to think and it is still up to students to figure out the details themselves.

3.3 Loops

Iteration is a challenging topic. Our observation is that students have trouble condensing actions into a compact loop format. The problem is not with loop syntax, but with how to abstract the sequence actions into a set of statements that will be executed repeatedly.

Our problem solving process helps with this abstraction. Figure 2 shows a correctly completed worksheet for loops. Similar to Figure 1, the problem description and the shaded areas are provided to students and the rest are blanks that students need to fill out. It involves the following steps: (1) Students describe the solution as a sequence of actions in plain English. (2) Then students specify a sequence of C++ statements that implement the actions identified in step 1. (3) In the next, key phase, students find the patterns among the sequence of C++ statements and abstract them into a set of C++ statements, which can be executed repeatedly to produce the same behavior as the sequence in step 2. Finally, students (4) determine how to stop the loop, and (5) determine how to set up the variables before the loop and the actions to perform after the loop. Students have freedom to choose a proper variable name for a data item as long as consistency is maintained throughout the worksheet.

Challenges: The problem solving processes and the accompanying worksheets presented above provide off-line scaffolding for modeling and coaching on how to construct programs to solve problems. They guide students and scaffold them in following a good analytical process. However, our classroom experience is that while some students can make good progress, some required additional help and some still didn't know what to do. For the last group, it is difficult to conduct one-on-one coaching during a class period. In addition, in our instructional experience, worksheets would not be effective at all outside the classroom for this group of students. Frequently these students failed to submit homework assignments due to their inability to use these worksheets independently.

Problem: If a student is a first-year and GPA is 3.0 and up, he gets *good starter* award. If GPA is 3.5 and up, he gets *dean's list* award.

Case Analysis

ID	Categorization of the Cases	Actions of the Cases
1	Student is a first-year and GPA is 3.0 or up	Student gets good starter award.
2	Student's GPA is 3.5 or up	Student gets dean's list award.

Exclusiveness/Order Analysis

Are the above cases exclusive to each other?	No
If no, please list the order of cases:	Order does not matter.

Logic Expressions and Actions

ID	Logic Expressions	Actions
1	classification==1 && gpa >= 3.0	cout << "Good Starter Award." << endl;
2	gpa >= 3.5	cout << "Dean's List Award." << endl;

Put the If Statement(s) Together

If the cases are exclusive, please use one multi-branch if-else statement. Otherwise, use independent if statement for each case.

```
if (classification == 1 && gpa >= 3.0) {
    cout << "Good Starter Award." << endl;
}
if (gpa >= 3.5) {
    cout << "Dean's List Award." << endl;
}
```

Figure 1: Pre-programming Analysis for Ifs

Problem: Calculate 2 + 4 + 6 + … + 10000 and display the result.

Describe the solution in sequence of actions (plain English):

```
Add 2 to sum
Add 4 to sum
…
Add 10000 to sum
```

Translate the above actions into C++ statements:

```
sum = sum + 2;
sum = sum + 4;
…
sum = sum + 10000;
```

Abstract into a set of same statements that can be repeatedly executed:

```
sum = sum + num;
num += 2;
sum = sum + num;
num += 2;
…
```

Condition that the loop should stop:

```
num > 10000
```

The condition that the loop should continue (reverse condition):

```
num <= 10000
```

Variables to set up before the loop:

```
sum = 0;   num =2;
```

Actions after the loop:

```
cout << "2+4+6+…+10000 = " << sum << endl;
```

Figure 2: Pre-programming Analysis for Loops

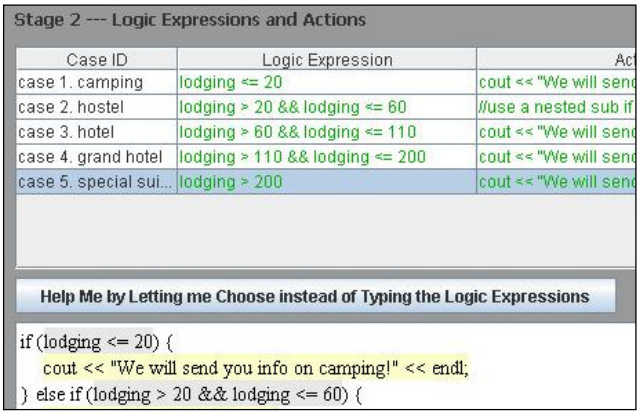

Figure 3: Snapshot of a Tutor Interface

4. THE CT COMPONENT

Our approach to address the challenge described at the end of Section 3 is to use Cognitive Tutors that interact with students within the framework of the cognitive apprenticeship learning model. The tutors provide the step-by-step support each student needs to complete each planning and programming task successfully. The tutor indicates whether each student action is correct or incorrect. If incorrect, students can request a hint as to why it is wrong. Some decisions, such as variable names and logical expressions, require students to type in the answers, but most decisions are multiple-choice. Figure 3 displays a snapshot of a part of the tutor interface during a step in solving a programming problem using if statements. The students can either type in the logic expressions or, after repeated failure, may request the tutor to present a multiple-choice menu from which to select a logic expression.

Program construction is divided into several stages. The if-statement tutor, for example, consists of the four stages. In stage (1), the student identifies the cases in a programming problem (selects English descriptions from menus). In stage (2), shown in Figure 3, the student enters logic expressions and actions for the cases. In stage (3) the student determines whether each logic expression can be simplified. The part of a logic expression that would be always true when it is evaluated can be safely removed. Logic expressions in a multi-branch if statement can often be simplified. Finally, in stage (4) students repeat stage 1-3 for if statements that will be nested in the top-level if statement. After students make correct answers for a stage, the tutor displays a partially completed program that incorporates the decisions the students have made for that stage. For the example in Figure 3, the logic expressions and actions the student enters at the top of the screen are incorporated into an actual if statement at the bottom of the screen. This way, a program is gradually constructed/refined based on students' decisions. Students will observe how their decisions in each step contribute to the construction of the program. After students finish the program, they can copy and paste the program into a standard IDE (e.g. Microsoft Visual Studio) to compile and test.

5. EFFECTIVENESS EVALUATION

These CAL and CAL-CT learning activities were incorporated into introductory programming courses at Shaw University and in this report we use student test scores to evaluate these interventions for each of the three programming topics across semesters (Fall 2007 – Spring 2010). For each of the three constructs, Variables & Basic Statements, Ifs, and Loops, all the students in the same semester used one of the learning approaches: No-CAL (the baseline), CAL-Only, or CAL-CT. Table 1 shows student assignment by semester.

Table 1: Student Assignment by Semester

	Basic Statements	**If**	**Loops**
Fall'07	CAL-Only	No-CAL	
Spring'08	CAL-Only	CAL-Only	No-CAL
Fall'08	CAL-Only	CAL-Only	CAL-Only
Spring'09	No-CAL	No-CAL	No-CAL
Fall'09	CAL-CT	CAL-CT	CAL-Only
Spring'10	CAL-CT	CAL-CT	CAL-CT

For each topic, the instructor introduced the programming construct in class and then students completed some program-tracing and syntax-debugging activities. At this point, a pretest was given to students. That is, at this point, students had learned the syntax of a programming construct but had not any programming experience for that construct. After the pretest, students worked on some programming problems in class. In the CAL-Only and CAL-CT conditions, the teacher demonstrated the use of the CAL worksheets in completing these example programs. Students then completed a homework assignment consisting of a set of problems in which they wrote programs.

- In the No-CAL condition, students completed the set of programming homework problems without any additional scaffolding.
- In the CAL condition, CAL worksheets were made available for the homework assignment and students were encouraged, but not required, to use the worksheets to help design the programs.
- In the CAL-CT condition, students completed a small set of homework problems and CAL worksheets were made available for that purpose. Students then completed a set of programming exercises in the CAL-CT environment. Note that there is no CT-Only approach for two reasons: (1) the classroom use of worksheets is in common to CAL CAL-CT by design, and (2) we wanted to conduct a within-student analysis of the "added value" in going from CAL worksheets to the CT environment, as discussed under section 5 "Learning Progress in the CAL-CT condition."

Following each homework assignment, the instructor reviewed the answers in class in all conditions. Finally, students completed a posttest in class. The test questions were all problem-based, asking students to write complete programs. We collected student test data over multiple semesters and tried to keep the test problems for each construct as similar as possible across semesters. We used self-developed and expert-inspected grading rubrics to evaluate student tests and generate test scores. Since our focus is problem solving, our rubrics give more weight (points) to students' approach to solving problems than to program syntax. Table 2 shows the number of students in each condition who participated in the study.

Table 2: Number of Students in Each Group

Condition	**Basic Statements**	**If Statements**	**Loops**
No-CAL	7	35	36
CAL-Only	55	35	19
CAL-CT	39	31	9

Figure 4 displays average posttest scores for the three learning conditions for each of the three programming topics and overall. As expected, students in the CAL-CT group scored higher than students in the other two conditions for each of the three topics and this overall result is significant by a non-parametric test of independent probabilities, $p < 0.04$.

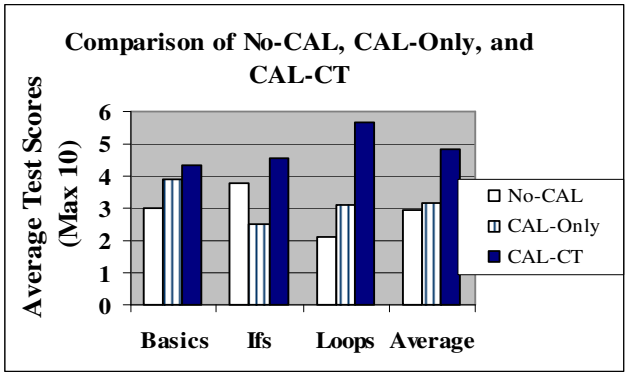

Figure 4: Posttest Comparisons among Three Groups

For the Loop construct, the differences among the groups were quite large. Posttest scores in the CAL-CT group were 82% higher than in the CAL-Only group, which in turn were 48% higher than the No-CAL baseline group. In a between-subject ANOVA, the main effect of learning condition is reliable, $F(2, 61) = 5.02$, $p < .01$. In pair-wise comparisons, the difference between CAL-CT and CAL-Only is reliable, $p < .04$, as is the difference between CAL-CT and No-CAL $p < .01$, while the difference between CAL-Only and No-CAL is not significant.

For the If construct, posttest scores for the CAL-CT group were 22% higher than in the No-CAL condition and 82% higher than in the CAL-Only condition. In a between-subject ANOVA, the main effect of group type is marginally significant, $F(2, 98) = 2.22$, $p < .12$, and the only significant pair-wise difference is between the CAL-CT and CAL-Only groups, $p < .05$. We believe that the close similarity of an example with the posttest solution led to the high posttest scores for Fall'07 (part of No-CAL), which caused the "skew" that No-CAL outperforms CAL.

Finally, for Variable & Basic Statements, the differences among the three groups were much smaller and in a between-subjects ANOVA, the main effect of group is not significant. In fact, the CAL-Only group for basic statements is not strictly CAL only because this group has used preprogramming analysis tutors as reported in [11]. These tutors are much simpler. They helped students fill out the preprogramming analysis worksheet without leading students to the final solutions (complete programs).

Learning Progress in the CAL-CT condition. In the CAL-CT condition, pretests and intermediate tests were used to examine how students made progress in learning. Students completed programming problems on these tests analogous to the posttest problems. For the Variables & Basic Statements construct, each student took a pretest before the homework assignment and completed an intermediate test after working on the initial set of problems with CAL worksheets and before beginning the CAL-CT problems. For the If construct, each student completed either a pretest or an intermediate test. For the Loop construct, each student completed an intermediate test.

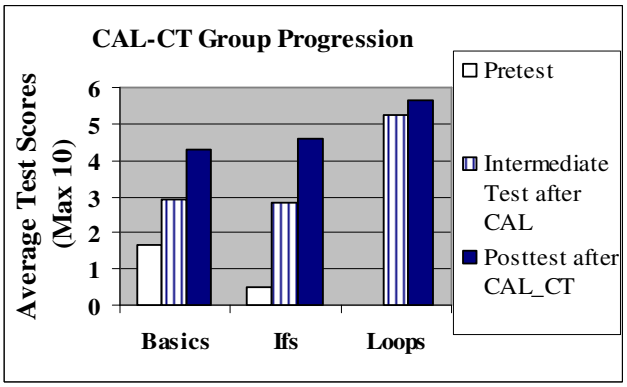

Figure 5: Student Performance Progression (CAL-CT)

Figure 5 displays the results of the pre-, intermediate and post tests. For both the Variables & Basic Statement constructs and the If construct, the learning activities with off-line CAL worksheets alone yielded substantial learning gains between the pretest and the intermediate test. This gain for Variables & Basic Statements is reliable in a paired-sample t-test, $t(20) = 2.75$, $p < .02$ and the gain for Ifs is marginally reliable in an independent-sample t-test, $t(26) = 2.61$, $p < .08$. For these two constructs, the Cognitive Tutor CAL-CT activities yielded substantial additional learning between the intermediate test and posttest. These respective gains were reliable in paired-sample t-tests, $t(35) = 3.34$, $p < .01$ and $t(19) = 2.62$, $p < .02$. In contrast, for the Loop construct, CAL-CT problem solving yielded little additional learning between the intermediate test, which followed the use of off-line CAL worksheets, and the posttest, and the difference between the two tests is not significant. This latter result is surprising since, as we've seen, the CAL-CT group posttest scores are reliably higher than the CAL-Only posttest scores (Figure 4).

Table 3: Student Surveys – Full or High Agreement

Survey Questions	No-CAL (28)	CAL-CT (24)
Felt motivated to come to every session.	57%	75%
Felt motivated to complete all hw assignments.	64%	79%
The instruction was very different from other courses.	67%	71%
The instruction was high quality.	68%	92%
The materials were excellent.	57%	88%
The course increased the excitement I have on learning C++ programming.	54%	71%
This course convinced me to stay a computer science major or to become on.	46%	50%
This course kept me or made me become interested in a CS career.	64%	71%
This was the best course I took this sem.	29%	33%

Table 3 shows student survey data collected during Spring 2009 (No-CAL) and Spring 2010 (CAL-CT). Students rated their agreement with each statement on a 5-point scale. Table 3 displays the percentage of students who "fully" or "highly" agreed with each statement. Clearly CAL-CT students tend to have more positive attitudes toward the course activities. A higher percentage of CAL-CT students than No-CAL students fully or highly agreed with every statement. The biggest differences between the two groups were for the 4th and 5th questions, "The instruction in this course was high quality," and "The course materials were

excellent," and both of these differences are significant at the .05 level in a z-test of the difference between two independent samples.

6. CONCLUSION AND DISCUSSION

CAL-CT combines two effective approaches for developing problem solving skills. The CAL-style preprogramming analysis allows students to model experts' analytical thinking and problem solving processes and helps them to mature as programmers. Online tutors give students individual attentions which are usually not available outside classrooms and keep them moving forward.

The CAL-CT Tutors are based on Java Swing technology and can be delivered to students easily online through either Java Applets or Java Web Start. Tutors are designed to be flexible enough to make it easy to incorporate new problems. Teachers can specify programming problems/solutions and tutor behaviors using text files without any programming.

We will add tutors for more programming concepts and develop adaptive problem delivery mechanisms to better fit individual student's needs. Our ultimate goal is to develop an engaging CAL-CT curriculum to help students master programming skills.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0837505.

Our thanks to Yingqi Wang for assistance in developing the tutors, Susan Rodger and Sung-Sik Kwon for their tutor-interface suggestions and Barry Nagle for designing student surveys.

8. REFERENCES

[1] Anderson, J. R., Conrad, F. G., and Corbett, A. T. 1989. *Skill Acquisition and the LISP Tutor*. Cognitive Science 13(4), 467-505.

[2] Anderson, J. R., Corbett, A. T., Koedinger, K. R., and Pelletier, R. 1995. *Cognitive Tutors: Lessons Learned*. J. of the Learning Sciences, 4(2), 167-207.

[3] Bloom, B. S. and Krathwohl, D. R. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners*. Handbook I: Cognitive Domain. New York, Longmans, Green.

[4] Bloom, B. S. 1984. *The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring*. Educational Researcher, 13(6), 4-16.

[5] Collins, A., Brown, J. S., Newman, S. E. 1989. *Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing and Mathematics*. In L. Resnick (Eds.), Knowing, Learning and Instruction, Essays in Honor of Robert Glaser. Erlbaum, Hillsdale, NJ.

[6] 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.

[7] Corbett, A. T., McLaughlin, M. S. and Scarpinatto, K. C. 2000. *Modeling student knowledge: Cognitive tutors in high school and college*. User modeling and user-adapted interaction, volume 10, 81-108.

[8] Fernandes, E. and Kumar, A. N. 2004. *A Tutor on Scope for the Programming Languages Course*. ACM SIGCSE Bulletin, volume 36, issue 1 (March 2004), 90 -93.

[9] Heller, P., Keith, R., and Anderson, S. 1992. *Teaching Problem Solving Through Cooperative Grouping. Part 1: Group versus Individual Problem.* American Journal of Physics, Volume 60, Issue 7 (July 1992), 627-636.

[10] Hristova, M., Misra, A., Rutter, M., and Mercuri, R. 2003. *Identifying and Correcting Java Programming Errors for Introductory Computer Science Students.* SIGCSE'03: Proceedings of the 34th SIGCSE technical symposium on Computer science education, 486-490.

[11] Jin, W. 2008. *Pre-programming Analysis Tutor Helps Students Learn Basic Programming Concept*. SIGCSE'08: Proceedings of the 39th SIGCSE technical symposium on Computer Science Education, 276-280.

[12] Koedinger, K. R. and Corbett, A. T. 2006. *Cognitive Tutors: Technology brings learning science to classroom*. In K. Sawyer (Ed.), the Cambridge Handbook of the Learning Sciences, Cambridge University Press, 61-78.

[13] Lane, H. and VanLehn, K. 2003. *Coached Program Planning: Dialogue-Based Support for Novice Program Design*. SIGCSE'03: Proceedings of the 34th SIGCSE technical symposium on Computer science education, 148-152.

[14] McCracken, M. et al. 2002. *A multi-national multi-institutional study of assessment of programming skills of first-year CS students*. SIGCSE Bulletin, Vol. 34, No. 1 (March 2002).

[15] Odekirt-Hash, E. and Zachary, J. L. 2001. *Automated Feedback on Programs Means Students Need Less Help from Teachers*. SIGCSE'01: Proceedings of the 32nd SIGCSE technical symposium on Computer Science Education, 55-59.

[16] Shaffer, S. C. Ludwig 2005. *An online programming tutoring and assessment system*. Inroads --- The SIGCSE Bulletin, Volume 37, Issue 2 (June 2005), 56-60.

[17] Soh, L. K. 2006. *Incorporating an Intelligent Tutoring System into CS1*. SIGCSE'06: Proceedings of the 37th SIGCSE technical symposium on Computer science education, 486-490.

[18] Song J. S., Hahn, S. H., Tak, K. Y. & Kim, J. H. 1997. *An intelligent tutoring system for introductory C language course*. Computers & Education, Volume 28, Issue 2 (February 1997), 93-102.