Int J Artif Intell Educ
DOI 10.1007/s40593-015-0091-7

ARTICLE

Do You Think You Can? The Influence of Student Self-Efficacy on the Effectiveness of Tutorial Dialogue for Computer Science

Joseph B. Wiggins[1] · Joseph F. Grafsgaard[1] · Kristy Elizabeth Boyer[2] · Eric N. Wiebe[1] · James C. Lester[1]

Abstract In recent years, significant advances have been made in intelligent tutoring systems, and these advances hold great promise for adaptively supporting computer science (CS) learning. In particular, tutorial dialogue systems that engage students in natural language dialogue can create rich, adaptive interactions. A promising approach to increasing the effectiveness of these systems is to adapt not only to problem-solving performance, but also to a student's characteristics. Self-efficacy refers to a student's view of her ability to complete learning objectives and to achieve goals; this characteristic may be particularly influential during tutorial dialogue for computer science education. This article examines a corpus of effective human tutoring for computer science to discover the extent to which considering self-efficacy as measured within a pre-survey, coupled with dialogue and task events during tutoring, improves models that predict the student's self-reported frustration and learning gains after tutoring. The analysis reveals that students with high and low self-efficacy benefit differently from tutorial dialogue. Student control, social dialogue, and tutor moves to increase efficiency, may be particularly helpful for high self-efficacy students, while for low self-efficacy students, guided experimentation may foster greater learning while at the same time potentially increasing frustration. It is hoped that this line of research will enable tutoring systems for computer science to tailor their tutorial interactions more effectively.

Keywords Self-efficacy · Tutorial dialogue · Computer science education

✉ Joseph B. Wiggins
jbwiggi3@ncsu.edu

[1] North Carolina State University, Raleigh, NC, USA

[2] University of Florida, Gainesville, FL, USA

Published online: 16 February 2016

Introduction

One-on-one tutoring has been shown to be more effective than conventional classroom settings (Bloom 1984). Due to the disparity between individualized tutoring and conventional classrooms, human tutoring has been the subject of considerable investigation (Chi et al. 2001; McKendree 1990; VanLehn 2011). Studies have examined the interaction between students and tutors from multiple standpoints: cognitive and affective outcomes (Boyer et al. 2008), the adaptive presentation of instructional material (D'Mello et al. 2009), motivational strategies (Lepper et al. 1993), and the exchange of rich natural language dialogue (Litman et al. 2009).

Development of tutorial dialogue systems presents many challenges, but these systems can currently engage with students in topic areas such as physics (Chi. M et al. 2010), electricity and electronics (Dzikovska et al. 2010), logic (Croy et al. 2007), and, as is the focus of this paper, computer science (Chen et al. 2011; Ezen-Can and Boyer 2015; Gerdes et al. 2012; Lane and VanLehn 2005; Lane et al. 2013; Vail et al. 2015).

It is generally acknowledged that students do not benefit uniformly from learning interactions with intelligent tutoring systems (VanLehn et al. 2007). While tutoring has a negligible impact on some students, others thrive and experience significant learning. Some disparity can be explained by modeling the student's interest in the subject, memory capacity, and attention abilities (D'Mello et al. 2009), but these differences do not capture all the variance. There has been a movement in the ITS community to approach the learning process from a more holistic standpoint. In addition to the cognitive aspects of learners, these systems consider learner characteristics such as motivation, metacognition and affect (du Boulay et al. 2010; Narciss 2013). In this article we examine one such characteristic, self-efficacy. There is evidence that self-efficacy is highly indicative of a student's self-regulatory abilities; the higher a student's self-efficacy, the better her self-regulatory abilities (Wang and Wu 2008). Adapting to learners' self-efficacy may be especially beneficial to students in fields in which there are well-documented social stigmas, such as computer science (DiSalvo et al. 2011; Koch 1994; Sadker 1999). In these situations, members of underrepresented groups can feel as though they do not belong (Anderson-Rowland et al. 2007) or that they have fixed aptitude that cannot be improved through study and practice (Dweck 2002). Students experiencing these feelings may easily become bored or disengaged—states that are associated with negative effects on learning (Pekrun et al. 2010).

This article examines the hypothesis that students with high or low self-efficacy benefit from different tutorial strategies to promote positive learning outcomes in the computer science domain. Models of tutorial interaction were built for students with high or low self-efficacy and compared to one another. The results reveal that learning and frustration are predicted by substantially different sets of tutorial interactions depending on students' incoming self-efficacy.

Background

A number of tutorial dialogue research projects have focused on computer science. An early system, ProPL, focused on planning in pseudocode (Lane and VanLehn 2005);

AutoTutor uses spoken dialogue to tutor basic computer literacy (Graesser et al. 1999); JavaTutor uses cognitive and affective dialogue, machine learned from human-human tutorial interactions, to interact with the student during the programming task (Vail et al. 2015; Ezen-Can and Boyer 2015); iList provides feedback as students learn about linked lists (Fossati et al. 2009).

Developing an effective intelligent tutoring system is a complex task, especially within a discipline that is dynamically changing such as computer science. There is debate regarding which programming languages are best to learn first; therefore, to be applicable to the variety of different forms computer science education can take, research must also make language-independent strides forward to stay relevant in the constantly evolving landscape (Rivers and Koedinger 2012). There is also the difficulty of assessing student programs for correctness, which is a serious challenge considering the variety of ways an answer could be wrong and the number of possible solutions to any given programming problem (Xu and Chee 2003).

In light of these challenges, research on AI systems for teaching computer science has made great progress facilitating learning. However, these systems have much room for improvement in adapting to individual differences. Self-efficacy is a particularly important characteristic, as it shapes students' beliefs in their ability to reach their learning goals. Self-efficacy influences not only how students approach problems, but also their coping behaviors during learning (Bandura 1977; Bernacki et al. 2015). There is also strong evidence that self-efficacy is highly indicative of a student's self-regulatory abilities (Wang and Wu 2008). Due to these factors, self-efficacy should be considered when students are facing difficult tasks, such as learning computer programming languages.

Some empirical work has begun to establish ways in which self-efficacy is influential in, and influenced by, tutoring for computer science. In an exploratory experiment on computer-mediated computer science tutoring for undergraduates, students with high self-efficacy made more declarative statements, or assertions, than students with low self-efficacy. Tutors paired with low self-efficacy students gave more negative feedback and made fewer acknowledgements than tutors paired with high self-efficacy students (Boyer et al. 2007). In a different study in the same domain (and on a subset of the corpus examined in this article), it was observed that self-efficacy for computer science (or CS-specific self-efficacy) and pre-test scores contributed highly to predictive models of normalized learning gain (Mitchell et al. 2013). It has also been discovered that tutors attempting to motivate students to increase their participation may have negative effects on those students, especially for students with low self-efficacy (Howley et al. 2012). Self-efficacy also appears to exhibit important relationships with nonverbal expressions of affect: students with low self-efficacy may use two-hands-to-face gestures significantly more frequently, which may indicate reduced focus on the task (Grafsgaard et al. 2013). There are also different facial expression predictors for self-efficacy between middle school and college-age students (Grafsgaard et al. 2015).

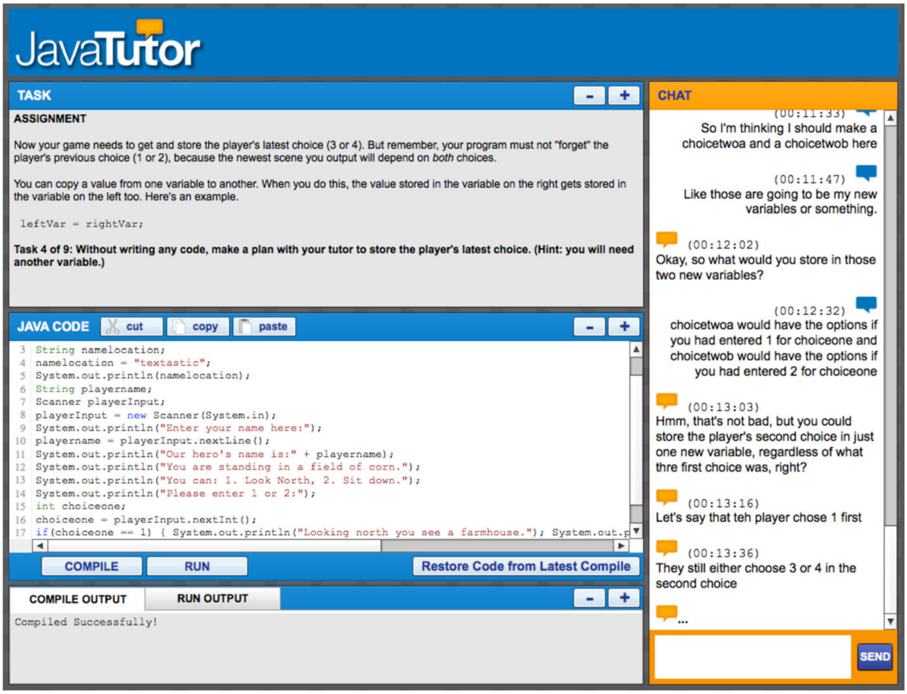
The study reported in this article builds upon this body of prior research in order to better understand the role that student self-efficacy has on cognitive and affective learning outcomes. The resulting models have important implications for how intelligent tutoring systems should choose to interact with students based on those students' incoming self-efficacy.

Tutorial Dialogue Corpus

The corpus consists of human-human computer-mediated tutorial dialogue for introductory computer science collected during the 2011–2012 academic year. The corpus was collected as part of the larger JavaTutor project, which aims to create a tutorial dialogue system that learns its behavior by modeling the strategies of human tutors.

In the human-human study under investigation, students ($N=67$) and human tutors interacted through a web-based integrated development environment (Mitchell et al. 2013). This environment presented a series of learning tasks to students, which led the students through the process of creating a text-based adventure game. Successful completion of the learning tasks required students to learn about and use variables, console I/O, conditionals including nested conditionals, and for loops. Students and tutors communicated through textual dialogue (Fig. 1). The tutors were experienced graduate (and in one case undergraduate) student teaching assistants. Of the 67 students who participated in the study, one is omitted from further analysis due to a missing post-test score.

The participants were university students in the United States with an average age of 18.6 years (SD = 1.3). They voluntarily participated in the study in exchange for course credit on a lab assignment in an introductory engineering course. No computer science knowledge was required for participants, and students with substantial self-reported prior programming experience were excluded from the study since its target audience was novices. Each student was paired with a tutor for a total of six sessions on different days, limited to forty minutes per session. This analysis focuses on the student-tutor



ASSIGNMENT

Now your game needs to get and store the player's latest choice (3 or 4). But remember, your program must not "forget" the player's previous choice (1 or 2), because the newest scene you output will depend on *both* choices.

You can copy a value from one variable to another. When you do this, the value stored in the variable on the right gets stored in the variable on the left too. Here's an example.

```
leftVar = rightVar;
```

Task 4 of 9: Without writing any code, make a plan with your tutor to store the player's latest choice. (Hint: you will need another variable.)

```
String namelocation;
namelocation = "textastic";
System.out.println(namelocation);
String playername;
Scanner playerInput;
playerInput = new Scanner(System.in);
System.out.println("Enter your name here:");
playername = playerInput.nextLine();
System.out.println("Our hero's name is:" + playername);
System.out.println("You are standing in a field of corn.");
System.out.println("You can: 1. Look North, 2. Sit down.");
System.out.println("Please enter 1 or 2:");
int choiceone;
choiceone = playerInput.nextInt();
if(choiceone == 1) { System.out.println("Looking north you see a farmhouse."); System.out.p
```

```
Compiled Successfully!
```

(00:11:33)
So I'm thinking I should make a choicetwoa and a choicetwob here

(00:11:47)
Like those are going to be my new variables or something.

(00:12:02)
Okay, so what would you store in those two new variables?

(00:12:32)
choicetwoa would have the options if you had entered 1 for choiceone and choicetwob would have the options if you had entered 2 for choiceone

(00:13:03)
Hmm, that's not bad, but you could store the player's second choice in just one new variable, regardless of what thre first choice was, right?

(00:13:16)
Let's say that teh player chose 1 first

(00:13:36)
They still either choose 3 or 4 in the second choice

...

Fig. 1 The JavaTutor tutorial dialogue interface for introductory Java programming

dialogue and coding events, however, recordings of the sessions also included webcam video, skin conductance recordings, and Kinect depth video (Grafsgaard et al. 2013, 2015; Hardy et al. 2013). There were a total of 29,840 task actions and dialogue moves corresponding to Session 1, which is the only session considered in this article. The set of possible interaction events, including student dialogue messages, tutor dialogue messages, and task actions are shown in Table 1.

Programming Task

The programming task that the student and the tutor were working on collaboratively is the development of a text-based interactive adventure game in Java. Students were guided through the basics of compiling and running, commenting, printing, declaring variables and gathering and storing input from the user. A sample student program is shown in Fig. 2. This student code will compile, but it displays the common error of forgetting to print an input prompt before reading from console input. The student, upon testing her code, will encounter this logic error.

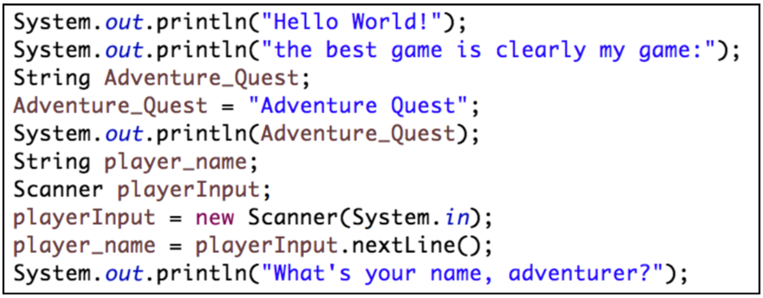
Self-Efficacy Measures

Before any tutoring and before the pre-test, a pre-survey was administered containing two questionnaires for self-efficacy: general and CS-specific. General self-efficacy is a

Table 1 List of task actions and dialogue moves

Task action	Description	Avg. instances per session
SESSIONPROGRESS	Tutor moved the session forward to the next task	16.03
TUTORMSG	Tutor sent a dialogue message to the student	88.21
STUDENTMSG	Student sent a dialogue message to the tutor	36.04
CODE+	Student has written code that is closer in terms of edit distance to the correct solution* than the prior code	60.63
CODE-	Student has written code that is farther in terms of edit distance from the correct solution* than the prior code	12.57
CODE0	Student has written code that is neither closer nor farther from the correct solution* in terms of edit distance than the prior code (e.g., changing string literals)	167.97
COMPILEBEGIN	Student clicked the COMPILE button	15.67
COMPILESUCCESS	Student’s code had no syntax errors and compiled successfully	12.06
COMPILEERROR	Student’s code had syntax errors and could not compile successfully	3.61
RUNCODE	Student clicked the RUN button	10.99
RUNSUCCESS	Student finished running the code with no errors	10.06
RUNTIMEERROR	Student encountered a runtime error while running the program	0.03
RUNSTOP	Student halted the currently active program by clicking STOP	0.76
INPUTSENT	Student entered console input to test program	10.73

* A correct solution is any code that a tutor deemed correct enough to move the session to the next task



```
System.out.println("Hello World!");
System.out.println("the best game is clearly my game:");
String Adventure_Quest;
Adventure_Quest = "Adventure Quest";
System.out.println(Adventure_Quest);
String player_name;
Scanner playerInput;
playerInput = new Scanner(System.in);
player_name = playerInput.nextLine();
System.out.println("What's your name, adventurer?");
```

Fig. 2 Sample (buggy) student code. This program awaits input before prompting the user for that input

measure of how students view their own ability to complete tasks and reach their goals. The higher a student's general self-efficacy is, the more a student believes in her ability to learn and grow academically. To measure student general self-efficacy, a scale developed by Chen et al. (2001) was used, which can be seen in Table 2.

CS-specific self-efficacy is the student's belief in her ability to learn and solve problems in the particular domain of computer science. To measure student self-efficacy for CS, a previously developed scale was modified for this use as part of the JavaTutor project was used (Wiebe et al. 2003) which can be seen in Table 3. Both measures of self-efficacy were calculated by taking the average of the student's responses (1: Strongly disagree; 2: Disagree; 3: Neutral; 4: Agree; 5: Strongly Agree). For reasons that will be described below, we utilize general self-efficacy for the analysis presented in this article.

Cognitive and Affective Outcome Measures

After Session 1, a post-session survey was administered to the students. This survey included questions to determine the affective outcomes of the lesson, including student frustration. The item being considered in the present study is Frustration Level (Hart and Staveland 1988), which was measured on a scale from 0 to 100 using the validated item, "How insecure, discouraged, irritated, stressed, and annoyed were you?"

To measure learning gain, an identical pre/post-test with 17 items was administered before and after the session. An example from the pre/post test for Session 1 is shown

Table 2 General self-efficacy questions

Student pre-survey general self-efficacy questions
1: I will be able to achieve most of the goals that I have set for myself.
2: When facing difficult tasks, I am certain that I will accomplish them.
3: In general, I think that I can obtain outcomes that are important to me.
4: I believe I can succeed at most any endeavor to which I set my mind.
5: I will be able to successfully overcome many challenges.
6: I am confident that I can perform effectively on many different tasks.
7: Compared to other people, I can do most tasks very well.
8: Even when things are tough, I can perform quite well.

Table 3 CS-specific self-efficacy questions

Student pre-survey CS-specific self-efficacy questions
1: Generally I have felt secure about attempting computer programming problems.
2: I am sure I could do advanced work in computer science.
3: I am sure that I can learn programming.
4: I think I could handle more difficult programming problems.
5: I can get good grades in computer science.
6: I have a lot of self-confidence when it comes to programming.

Table 4. No feedback was given to the students about how they performed on the pre/post-tests. Normalized learning gain was computed using the following formula if the posttest score was greater than the pretest score:

$$Normalized\ Learning\ Gain = \frac{Posttest-Pretest}{1-Pretest}$$

Otherwise, normalized learning gain was computed as follows (pretest>posttest):

$$Normalized\ Learning\ Gain = \frac{Posttest-Pretest}{Pretest}$$

The formula was derived from the work of Marx and Cummings (2007). This normalized learning gain calculations includes an adjustment to avoid division by zero, appearing in cases in which the student has a perfect pretest score. The items on the pre/post-test involved matching definitions, ordering code statements, and explaining what code fragments do. All of these were presented within multiple-choice formats.

Code Quality Measure

The computer science task focused on students developing a piece of code and evaluating it with their tutor. Therefore, it is important to analyse the students' code progress throughout the lesson. To measure a student's code quality as the session unfolded, the student's code was tokenized after any 1.5 seconds of code inactivity. Then, the tokenized code was compared to the code that the student had written at the

Table 4 Portion of a matching question from the pre/post-test administered to students

Next to each phrase in the right column, type the letter of the matching phrase from the left column that completes the sentence	
A) Makes space in the computer to store a value	Compiling a program __________
B) Translates human-readable code into machine language	A comment __________
C) gets information from the user to the program	An output statement __________
[remaining items omitted for brevity]	[remaining items omitted for brevity]

end of the lesson. By using the Levenshtein Distance formula, a token-level minimum edit distance algorithm, we produced the high-level tags of CODE+, CODE-, and CODE0 as described in Table 1.

Investigating Self-Efficacy

This study aims to identify differences in effective tutor interactions with students of high versus low self-efficacy. However, the two measures of self-efficacy have different theoretical underpinnings: for example, CS-specific self-efficacy, at first blush, might seem to be more strongly associated with outcomes of tutoring for computer science than general self-efficacy. On the other hand, general self-efficacy may be more stable over time than CS-specific self-efficacy (or any domain-specific self-efficacy; Chen et al. 2001). To investigate this, the measures of general self-efficacy and CS-specific self-efficacy were analysed in our corpus. Not surprisingly, across the students in this study, general self-efficacy and CS-specific self-efficacy were positively correlated ($r = 0.33$; $p = 0.0062$). On average, students reported a general self-efficacy of 4.139 out of a possible 5, with a standard deviation of 0.407. For CS-specific self-efficacy, the average was 3.240 out of a possible 5, with a standard deviation of 0.697. Figure 3 displays the spread of data for the self-efficacies.

It should be noted that the general self-efficacy reported was high on average compared to the range of its scale (1–5). This high self-efficacy across the sample of students under consideration here is likely due to the fact that the sample consists of college-level students enrolled at a large research university, and we would expect them to report higher self-efficacy measures than the general population. Although a further discussion of gender differences is beyond the scope of this article, Fig. 4 shows the breakdown between female and male general self-efficacy scores. The results on self-efficacy should be interpreted in light of the fact that gender (and likely many other

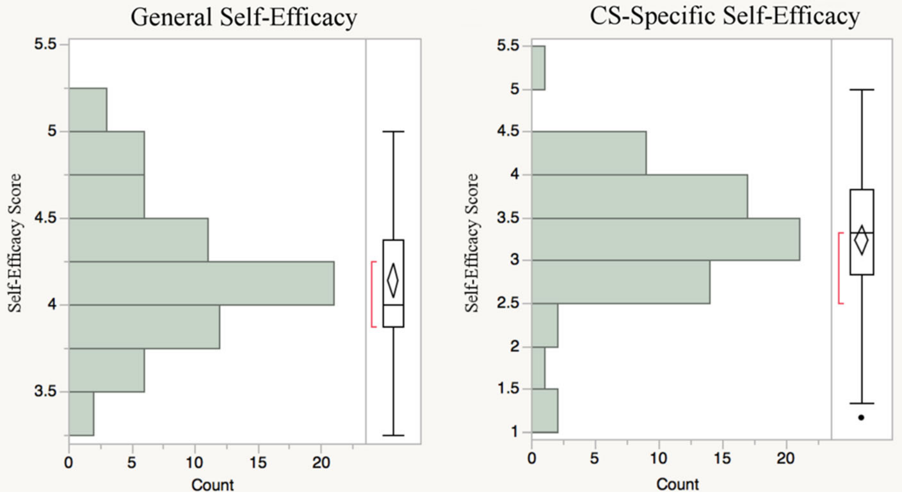


Fig. 3 Histograms of General Self-Efficacy (*left*) and CS-Specific Self-Efficacy (*right*)

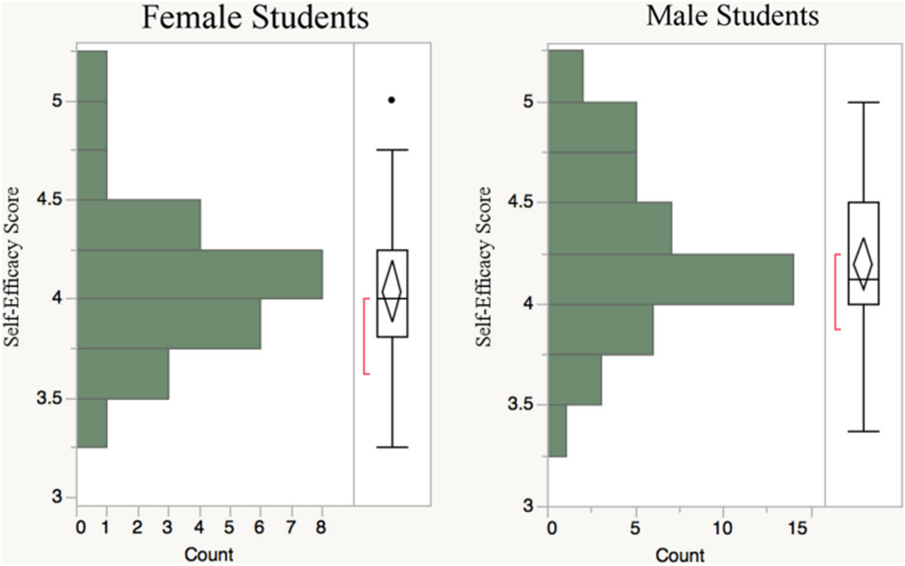


Fig. 4 Histograms of the General Self-Efficacy for Females (*left*) and Males (*right*)

student characteristics) can significantly influence the structure and content of tutorial dialogue.

Correlations with Learning Outcomes

Because general self-efficacy and CS-specific self-efficacy were correlated with one another, to proceed in the analyses we selected one measure of self-efficacy. To select between the two we computed their correlation with the outcomes of interest. As shown in Table 5, general self-efficacy was significantly correlated with Normalized Learning Gain ($r = 0.34$; $p = 0.006$) and Frustration ($r = -0.33$; $p = 0.005$). The mean Normalized Learning Gain for students in the low general self-efficacy group is 0.34 (st. dev. = 0.33), while the mean among students in the high general self-efficacy group is 0.50 (st. dev. = 0.29).

This relationship is stronger than that of CS-specific self-efficacy, so general self-efficacy score will be the measure used to represent the concept of student self-efficacy. Reflecting upon the student population in this study, it is not surprising that general self-efficacy is more informative than CS-specific self efficacy because these students were total novices (recall that any students who reported prior computer science experience were intentionally not included in this study). Novices may be much less able to judge their self-efficacy in a domain with which they have no experience, which is the case with these students and computer science.

Predictive Models

In order to explore differences in learning gain and frustration between students of high and low self-efficacy, we divided the students into two mutually exclusive bins by the

Table 5 Self-efficacy correlations with student outcomes

	General self-efficacy		CS-Specific self-efficacy	
	r	*p*	*r*	*p*
Pretest	−0.1795	0.1493	0.2569	0.0373
Normalized learning	0.3406	0.0055	0.2550	0.0403
Frustration	−0.3317	0.0051	−0.1529	0.2203

median general self-efficacy score. For the remainder of this paper we simply refer to general self-efficacy as self-efficacy. The data utilized are drawn from the first of six sessions in the human-human JavaTutor study. Of the 67 students who participated in the study, one is omitted from further analysis due to a missing post-test score. The remaining students ($N=66$) were split into a high self-efficacy bin ($N=32$) containing the students whose self-efficacy was above the median of 4 on a scale of 1 to 5. The low self-efficacy bin ($N=34$) contains those students who reported a self-efficacy equal to or below the median. This median split was selected because there is no theoretically established threshold for "high" and "low" self efficacy; therefore, we use the sample of students to situate each student as "high" or "low" with respect to all other students in the sample. As we discuss later in the Limitations section, a drastically different sample of students would be unlikely to maintain stability of these groups of "high" and "low" self-efficacy students.

Using the detailed logs of tutoring events (Table 1), we constructed a sequential representation of each tutoring session. We then extracted bigrams (pairs of adjacent interaction events) across those sequences. An interaction sequence (e_1, e_2, e_3, …, e_n) generates the following bigrams: (e_1, e_2), (e_2, e_3), …, (e_{n-1}, e_n). Using these extracted bigrams we computed the relative frequency of each bigram over the session, which controls for sessions of varying lengths.

Bigrams were selected as the unit of analysis for two reasons. First, compared to unigrams (single events considered in isolation), bigrams provide additional context by capturing the preceding event. Additionally, in contrast to higher order n-grams such as trigrams and 4-grams, bigrams avoid issues of sparsity that arise when considering these higher order n-grams. Bigrams are widely used in tutorial dialogue research as the unit of analysis (Forbes-Riley and Litman 2005; Boyer et al. 2009).

In order to model how these bigrams, which represent aspects of the tutorial interactions, are associated with the outcomes of learning gain and frustration, we built multiple regression models. The bigram relative frequencies were provided to the models as predictors, and separate models were built to predict normalized learning gain and frustration as response variables. Because the number of features is large—there is one feature for each bigram— we performed feature selection using model averaging, a technique that creates regression models for all possible combinations of predictor variables and then selects the most predictive variables using the average coefficient estimate from models with one, two, or three predictive variables (Symonds and Moussalli 2010). For example, if we only had four features, model averaging across models with one, two, or three predictive variables would generate the following: 4 models with one feature, 6 models with two features, and 4 models with three features (i.e., all possible combinations for each number of features). The coefficient estimates for each feature would then be averaged across the models (i.e., one average coefficient estimate per feature), with the top 20 features selected based on the average coefficient estimates.

Feature selection using model averaging was performed using JMP, then multiple regression models were also constructed in JMP. Using the top 20 features, forward stepwise feature selection was performed with leave-one-student-out cross-validated R^2 value as the selection criterion. If leave-one-student-out cross-validated R^2 value did not increase more than a specified epsilon value, the variable was not added to the model. This approach varies slightly from a traditional stepwise regression model, which inserts a variable if the p-value for that variable's coefficient is smaller than a threshold. This resulted in two R^2 values per model: one value to report the performance of the model constructed using all data, and a second value to report the average R^2 across the leave-one-out cross-validated models, which provides approximate evidence of generalizability.

The next three sections examine the resulting predictive models of learning gain and frustration built for all students, and then built separately for students of high and low self-efficacy.

All Students

The first set of models we will consider are those built on data from all students ($N = 66$) to predict normalized learning gain and standardized frustration. These models will allow us to compare bigrams that are predictive of learning and frustration in the full set of students versus the high and low self-efficacy groups. We hypothesize that the models that specialize by self-efficacy group will explain a greater proportion of the variance in outcomes than the model learned on all students.

All Students | Predicting Normalized Learning Gain

As shown in Table 6, two bigrams show a positive relationship with normalized learning gain in the overall group of students. First, a run success followed immediately by starting another compilation (RUNSUCCESS_COMPILEBEGIN) may indicate that the student is repeatedly testing her program, which is likely productive in uncovering code bugs as well as misconceptions held by the student. (Note that these novices may plausibly have held a separate, unrelated misconception that they needed to compile

Table 6 Model for learning gain built on all students ($R^2 = 0.353$; $r = 0.594$)

Normalized learning gain =	p
2.02 * RUNSUCCESS_COMPILEBEGIN	0.0080
0.64 * COMPILEBEGIN_COMPILESUCCESS	0.0245
−8.51 * TUTORMSG_RUNSTART	0.0024
−0.05 (intercept)	
RMSE = 0.277 Standard Deviations **Leave-One-Out Cross Validated $\mathbf{R^2}$** = 0.176	

before running the code each time, which may be influential in this bigram's appearance in the model.) The second bigram that significantly predicts learning is a compile attempt followed by compile success (COMPILEBEGIN_COMPILESUCCESS). This bigram indicates syntactically correct generation of programming code, a step toward completing the task and learning the related material.

The model selected one bigram that was negatively correlated with normalized learning gain: a tutor message followed by the student running the code (TUTORMSG_RUNSTART). These bigrams were often the tutor instructing the student to run the code, a didactic mode of tutoring that has previously been observed to be less effective (Mitchell et al. 2013).

Table 7 displays an excerpt illustrating some of the bigrams that were selected in the model in Table 6. In this excerpt the student improves the code quality and then compiles. After a superficial code edit, the tutor begins to explain some elements of the code in more detail to the student and then instructs the student to test the code.

All Students | Predicting Frustration

Next we turn our attention to the model built on all students to predict frustration. As shown in Table 8, one bigram was negatively correlated with frustration: moving to the next task followed by a superficial code edit (SESSIONPROGRESS_CODE0). Superficial code edits do not substantively change the program's functionality—for example, adding or editing comments. However, comments may have been used by students to organize their thoughts, and it is possible that this type of "scratch work" helped to reduce frustration.

Table 7 Excerpt containing bigrams identified by model for learning gain on all students

Excerpt 1	
Student:	[CODE+]
	[COMPILEBEGIN]^
	[COMPILESUCCESS]^
	[CODE0]
Tutor:	*On line 9, two things are happening*
	1) java stops at nextLine() and waits for user input.
	2) whatever is typed is put into (assigned) to PlanerName.
	Okay?
Student:	*Okay*
Tutor:	*Try it.**
Student:	[RUNSTART]*
	[INPUTSENT]
	[RUNSUCCESS]
Tutor:	*It worked as you wrote it*

The bigrams positively associated with learning gain are marked with ^ and those that are negatively associated are marked with *

Table 8 Model for frustration built on all students ($R^2 = 0.620$; $r = 0.787$)

Standardized frustration level =	*p*
−1.31 * SESSIONPROGRESS_CODE0	0.0253
108.64 * CODE + _ INPUTSENT	0.0345
6.40 * COMPILESUCCESS_STUDENTMSG	0.0001
6.93 * INPUTSENT_STUDENTMSG	0.0003
10.82 * TUTORMSG_CODE-	0.0297
1.38 * CODE-_CODE+	0.0055
1.87 * TUTORMSG_TUTORMSG	0.0163
−0.81 (intercept)	
RMSE = 0.664 Standard Deviations **Leave-One-Out Cross Validated** $\mathbf{R^2}$ = 0.369	

There were also six bigrams selected which were positively correlated with frustration. For example, tutor messages followed by negative code changes (TUTORMSG_CODE-) or another tutor message (TUTORMSG_TUTORMSG) were both predictive of increased frustration. These may correspond to a particularly didactic mode of tutoring and unsuccessful code modifications, respectively. As another example, successful compile followed by a student message (COMPILESUCCESS_STUDENTMSG) was also positive associated with frustration. We see that students who are progressing rapidly on the task will compile and then run. Compiling and then messaging the tutor likely indicates that the student is not sure what to do next, which is associated with increased frustration.

In Tables 9 and 10, excerpts with some of these bigrams are displayed. In Table 9, a student has successfully compiled the code and then the tutor suggests commenting out a line of code which will cause the code not to compile. The tutor then instructs the student to compile the code to see the error. In Table 10, the student successfully compiles the code and then asks the tutor to confirm correctness. The tutor tells the student to run the code and see that without adding new print statements, nothing

Table 9 Excerpt containing bigrams identified by model for frustration on all students

Excerpt 2	
Student:	[COMPILEBEGIN]
	[COMPILESUCCESS]
Tutor:	Very good. Any questions?
Student:	*I think im good. :)*
Tutor:	*Try commenting out line 4^ that is, make it begin with a //^*
Student:	[CODE-]^
	[CODE-]
Tutor:	*Now compile*
Student:	[COMPILEBEGIN]
	[COMPILEERROR]
Tutor:	*Make sense?^ you must do more than just declare^ a string variable^*

(* denotes negative correlation; ^ denotes positive correlation)

Table 10 Excerpt containing bigrams identified by model for frustration on all students

Excerpt 3	
Student:	[COMPILEBEGIN]
	[COMPILESUCCESS] okay, is that right?
Tutor:	*Good.*
	Try running
Student:	[RUNSTART]
	[RUNSUCCESS]
Tutor:	*nothing changed bc we did not add any print lines.^*
	*ready?^*
Student:	Yes
Tutor:	[SESSIONPROGRESS]*
Student:	[CODE0]*
	[CODE+]

(* denotes negative correlation; ^ denotes positive correlation)

different was printed to the console. They then move on to the next task and the student begins by making a superficial code edit.

High Self-Efficacy Students

Having seen the models for predicting learning gain and frustration in the overall group of students, we now turn our attention to more specialized models built to predict these outcomes for only students with high self-efficacy ($N = 32$). Recall that these students were grouped because their self-efficacy was above the sample median of 4 on a scale from 1 to 5. High self-efficacy students are those with a strong belief in their ability to reach their learning goals. In this data set, higher self-efficacy is associated with higher learning gains and decreased frustration. However, we hypothesize that the events during tutoring play an important role on learning gain and frustration. We examine the learned predictive models to explore that hypothesis.

High Self-Efficacy Students | Predicting Normalized Learning Gain

As shown in Table 11, for students with high self-efficacy, one bigram is positively predictive of learning gain: a tutor message followed by the student stopping a running program (TUTORMSG_RUNSTOP). These messages were often the tutor explaining something that is not correct in the student's program, and the student may stop the run in order to correct the code and then run again later.

Three bigrams were selected for having a negative relationship with learning gain: tutor message followed by a run success (TUTORMSG_RUNSUCCESS) or by a superficial code edit (TUTORMSG_CODE0), and run success followed by an increase in code quality (RUNSUCCESS_CODE+). These bigrams highlight the fact that a tutor proactively leading the student is not as helpful for high self-efficacy students, whereas independent work within the student's ability may foster learning. These

Table 11 Model of learning gain for high self-efficacy students ($R^2 = 0.716$; $r = 0.846$)

Normalized learning gain =	p
12.41 * TUTORMSG_RUNSTOP	0.0274
−32.26 * TUTORMSG_RUNSUCCESS	0.0445
−6.46 * RUNSUCCESS_CODE+	0.0053
−2.91 * TUTORMSG_CODE0	0.0066
0.61 (intercept)	
RMSE = 0.186 standard deviations **Leave-One-Out Cross-Validated $\mathbf{R^2}$** = 0.319	

bigrams highlight the tutor's actions causing decreased learning by interfering with the student's exploration of the session.

In Tables 12 and 13, excerpts containing bigrams identified for their association with learning gain in high self-efficacy students are displayed. In Table 12, the tutor instructs the student to make a superficial code edit and then tells the student to stop his/her current testing in order to attempt to compile the new code. In Table 13, the student successfully compiles the code and then starts running the code. The running code is waiting for user input and the tutor asks the student to identify the line of code that is causing the wait. The student enters input and then correctly identifies and modifies the logic error in the code.

High Self-Efficacy Students | Predicting Frustration

The previous models suggest that student control may foster learning for high self-efficacy students. We next explore predictors of frustration for these students. As shown in Table 14, one bigram was selected for a negative association with frustration: student message followed by run start (STUDENTMSG_RUNSTART). As seen in the excerpt in Table 12, this is often acknowledgment of tutor moves before running the program.

Table 12 Excerpt containing bigrams identified by model for learning gain on high self-efficacy students

Excerpt 4	
Tutor:	*So read the TASK and do Task 3**
Student:	[CODE0]*
Tutor:	*Good**
Student:	[CODE0]*
Tutor:	*ready?*
Student:	*i typed the comment and now do i hit compile*
Tutor:	*Try it*
	*Then run it^*
Student:	*[RUNSTOP]^*
	[COMPILEBEGIN]
	[COMPILESUCCESS]

(* denotes negative correlation; ^ denotes positive correlation)

Table 13 Excerpt containing bigrams identified by model for learning gain on high self-efficacy students

Excerpt 5	
Student:	[COMPILEBEGIN] [COMPILESUCCESS] [RUNSTART]
Tutor:	What line is it waiting on?
Student:	*[INPUTSENT]* *[RUNSUCCESS]** *[CODE+]**

(* denotes negative correlation; ^ denotes positive correlation)

Prior research has shown that high self-efficacy students make more acknowledgements during dialogue, and these social moves may indicate rapport with the tutor that is helpful in mitigating frustration (Boyer et al. 2007).

Two bigrams with a positive association with frustration were a student message followed by sending input to the running program (STUDENTMSG_INPUTSENT), which happened during program testing, and a code decrease in quality followed by a compilation attempt (CODE-_COMPILEBEGIN). Also positively associated with frustration were the student stopping the running code and then the tutor advancing to the next task (RUNSTOP_SESSIONPROGRESS) and a successful run followed by an increase in code quality (RUNSUCCESS_CODE+); both of these indicate positive progress toward completing the task.

In Table 15, an excerpt containing bigrams associated with frustration in high self-efficacy students is displayed. In this excerpt, the tutor is engaging the student in a dialogue about what is really happening when the code is compiled before the student tests his/her compiled code.

Low Self-Efficacy Students

The final grouping of students we consider are those with low self-efficacy ($N = 34$), those who reported a self-efficacy of 4 or lower on a scale from 1 to 5. Low self-efficacy students are those with a less strong belief in their ability to reach their learning goals. In

Table 14 Model of frustration for high self-efficacy students ($R^2 = 0.827$; $r = 0.909$)

Standardized frustration level =	*p*
−70.66 * STUDENTMSG_RUNSTART	<0.0001
43.58 * STUDENTMSG_INPUTSENT	<0.0001
4.00 * CODE-_COMPILEBEGIN	<0.0001
1.32 * RUNSTOP_SESSIONPROGRESS	<0.0001
7.13 * RUNSUCCESS_CODE+	0.0017
−0.66 (intercept)	
RMSE = 0.220 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.431	

Table 15 Excerpt containing bigrams identified by model for frustration on high self-efficacy students

Excerpt 6	
Tutor:	*What you're doing is actually complex*
	What do you think a machine language is like?
Student:	*Well to be honest I have no idea haha but it should be pretty straightforward*
Tutor:	*Yeah*
	Computers actually operate on 1's and 0's
	So, the Java compiler takes what you see in the Java Code panel and translates it to that
	[SESSIONPROGRESS]
Student:	*Alright, cool**
	*[RUNSTART]**

(* denotes negative correlation; ^ denotes positive correlation)

the JavaTutor data set we see that having a lower self-efficacy is related to decreased learning gains and higher frustration. The models we examine next examine what events within tutoring predict learning and frustration for lower self-efficacy students.

Low Self-Efficacy Students | Predicting Normalized Learning Gain

As shown in Table 16, there were three bigrams associated with increased learning: a student running his/her code and then increasing the code quality (RUNSTART_CODE+), finishing a run successfully being followed by increasing the code quality (RUNSUCCESS_CODE+) and a run stop being followed by a student utterance (RUNSTOP_STUDENTMSG). All these bigrams involve running the code. Interestingly, it is possible that the benefit to learning in these instances is coming from the student learning by doing, which may be particularly impactful for lower self-efficacy students, or from discussing what has just been done. The only bigram associated with decreased learning was a compile success followed by a tutor message (COMPILESUCCESS_TUTORMSG).

In the excerpt in Table 17, a low self-efficacy student successfully compiles and then begins running the code. However, after seeing what is printed to the screen, the student stops the code and asks the tutor for advice, correctly identifying the logic error in the code. The tutor then tells the student to try out the idea, and the student corrects the

Table 16 Model of learning gain for low self-efficacy students ($R^2 = 0.603$; $r = 0.777$)

Normalized learning gain =	*p*
9.27 * RUNSTART_CODE+	0.0194
6.04 * RUNSUCCESS_CODE+	0.0126
0.72 * RUNSTOP_STUDENTMSG	0.0047
−0.88 * COMPILESUCCESS_TUTORMSG	0.0070
0.62 (intercept)	
RMSE = 0.239 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.158	

Table 17 Excerpt containing bigrams identified by model for learning gain on low self-efficacy students

Excerpt 7	
Student:	[COMPILEBEGIN] [COMPILESUCCESS] [RUNSTART] [RUNSTOP]^ *Do I need to change the order to make the prompt before the input?^*
Tutor:	*Try it.*
Student:	[CODE+]

(* denotes negative correlation; ^ denotes positive correlation)

problem. In Table 18, a low self-efficacy student is making adjustments to the code and the tutor tells her that she is running out of time. The tutor suggests that they compile the code, which the student does successfully. Rather than give the student the opportunity to test the program in a hands-on way, the tutor describes what will happen and then proceeds to the next task. Consistent with previous observations, this short-circuit of hands-on practice may not be productive for students with low self-efficacy.

Low Self-Efficacy Students | Predicting Frustration

As shown in Table 19, the only bigram associated with decreased frustration for low self-efficacy students is a code decrease in quality followed by a compile attempt (CODE-_COMPILEBEGIN). This bigram indicates that the student may be checking code changes frequently, an iterative approach that may help mitigate frustration. On the other hand, a decrease in code quality followed by an increase in code quality (CODE-_CODE+), as well as an increase in code quality followed by input to a running program (CODE+_INPUTSENT), are associated with increased frustration. Interestingly, these two bigrams indicate independent steps taken toward completing

Table 18 Excerpt containing bigrams identified by model for learning gain on low self-efficacy students

Excerpt 8	
Student:	[CODE0] [CODE+]
Tutor:	*Ok. We are running out of time.* *Just compile your code for now.* [COMPILEBEGIN] [COMPILESUCCESS]*
Tutor:	*Alright. Your game will now prompt the player for her name and then display the input field.** *In interest of time, I am going to skip past the next task.* [SESSIONPROGRESS]

(*denotes negative correlation; ^ denotes positive correlation)

Table 19 Model of frustration for low self-efficacy students ($R^2 = 0.744$; $r = 0.863$)

Standardized frustration level =	p
−9.95 * CODE-_COMPILEBEGIN	0.0192
9.90 * COMPILESUCCESS_STUDENTMSG	<0.0001
158.08 * CODE + _INPUTSENT	0.0084
13.65 * TUTORMSG_CODE+	0.0350
2.56 * CODE-_CODE+	0.0072
−1.36 (intercept)	
RMSE = 0.734 standard deviations **Leave-One-Out Cross-Validated R^2** = 0.374	

the task without tutor intervention, which are in the same spirit as the bigrams that predicted increased learning. However, these predict increased frustration.

In Table 20, an excerpt that has bigrams associated with frustration in low self-efficacy students is displayed. In the excerpt, a low self-efficacy student is having trouble getting his/her code to compile and the tutor gives some advice, which the student initially misunderstands, but then when the tutor follows up, the student is able to make the correction.

Discussion

As we move toward creating highly effective tutoring systems to support computer science learning, it is important to consider the possibility that student characteristics

Table 20 Excerpt containing bigrams identified by model for frustration on low self-efficacy students

Excerpt 9	
Student:	[COMPILEBEGIN]
	[COMPILEERROR]
Tutor:	*The carat points to the error.*
Student:	[CODE-]^
	[CODE+]^
	[COMPILEBEGIN]
	[COMPILEERROR]
Tutor:	*Compare your line with example*^
	[CODE+]^
	[CODE0]
	[CODE0]
	[CODE0]
	[COMPILEBEGIN]
	[COMPILESUCCESS]

(* denotes negative correlation; ^ denotes positive correlation)

such as self-efficacy influence the effectiveness of tutorial interventions. We hypothesized that self-efficacy has a substantial influence on learning gain and frustration during one-on-one tutorial dialogue for introductory computer science. The models that we have presented provide support for this hypothesis.

One important consideration for the models is the extent to which they were able to explain the variance in the outcomes of interest. The All-Students models for frustration and learning achieved leave-one-student-out cross-validated R^2 of 0.35 and 0.62, respectively. In comparison, the models learned for High Self-Efficacy students achieved 0.71 and 0.83, while the models for Low Self-Efficacy students achieved R^2 of 0.60 and 0.74. In both cases, the specialized model outperformed the all-students model for explaining the outcome variables, despite having less data for training. This confirms that self-efficacy is an important consideration when modeling the effectiveness of tutorial dialogue for computer science.

For learning gain, we see important differences in the predictors for students with high self-efficacy and low self-efficacy. For students with high self-efficacy, positively associated with learning were tutor messages followed by the student stopping a running program. This move may have improved the efficiency of learning because rather than having the student complete a full test of the program to uncover a remaining bug, the tutor chose to proactively discuss it. High self-efficacy students may be able to keep up with this “shortcut” toward addressing errors, whereas the approach may not be as effective for low self-efficacy students. In contrast, positive predictors of learning for low self-efficacy students all involved running the program followed by either code improvements or a student dialogue move, but not any tutor dialogue moves. In fact, tutor dialogue moves only appear in one bigram for low self-efficacy students, and that is when the tutor chooses to send a message immediately after the student compiles successfully. That bigram is negatively associated with these students’ learning, suggesting that the importance of timing for low self-efficacy students may be greater than for high self-efficacy students where tutor messages do not appear in the second half of any significant bigram for predicting learning.

For frustration, there are also important differences in the predictors for students with high self-efficacy and low self-efficacy. For high self-efficacy students, we see decreased frustration with student messages followed by starting a code test, which often involved the student acknowledging tutor feedback and then testing the program. For low self-efficacy students, student dialogue messages only appeared once in the predictors for frustration, and that was following a compile success. In that case the messages were associated with more frustration. This situation illustrates an important possibility regarding natural language dialogue and self-efficacy: students with low self-efficacy may opt not to make non-essential dialogue moves, such as acknowledgments, as frequently as students with high self-efficacy, and there are two compelling possible reasons for this. One is that non-essential dialogue may represent extraneous load on a student who is already challenged by the task. A second is that low self-efficacy may also go hand in hand with a decreased sense of “standing” while conversing with a tutor, a phenomenon that computer science educators may observe in their classrooms by the hesitation of less confident students to ask questions or engage in classroom dialogue.

Finally, it is important to note a nuance of frustration. While it seems at first blush to be something to avoid, research has demonstrated that some degree of healthy frustration supports learning (D’Mello et al. 2014). Indeed, we saw that bigrams representing independent student work were associated with both higher learning and higher frustration, particularly for students with low self-efficacy. This is a different perspective on the cognitive-affective tradeoff (which suggests that in order to improve learning, affective considerations may have to take a backseat). The nature of the cognitive-affective tradeoff may depend upon student characteristics.

Limitations While this article provides insight to the relationship between self-efficacy and student learning outcomes during computer science learning, it is important to consider the limitations of the study. Firstly, we note limitations regarding the measured self-efficacy in the JavaTutor study. While self-efficacy could range from 1.0 to 5.0, the range in the sample considered here was only 3.25 to 5.0. These high reported self-efficacy values are not surprising for a sample of college freshmen at a large research university. In a sample with different students, the division between high and low self-efficacy would be at different point, and may change the phenomena observed in these groups. Additionally, we note that the present study treats self-efficacy as stable: students were grouped based on answers to the survey which was administered prior to tutorial interaction. It is possible that self-efficacy may change during learning, and these changes may affect students' behaviour (Bernacki et al. 2015). It is also important to note that this work does not consider the differences between tutors and their individual effectiveness. Such analysis has been addressed in previous work and is known to play a role in the student's learning outcomes (Mitchell et al. 2013).

Conclusion

As the computer science education community moves toward adaptive support for individual learners, it is crucial for intelligent learning environments to consider not only students' knowledge and skill, but also motivational factors driven by affect. Self-efficacy, students' beliefs in their own abilities, may have deep and far-reaching implications for computer science learning. Students' level of self-efficacy may, in particular, influence the types of adaptive support that are most effective.

This article has examined a data set of human-human tutoring for introductory computer science using the Java programming language. We have investigated predictive models of learning and frustration for students with high and low self-efficacy, finding that different interactions were predictive of these outcomes for the two groups. Student control, social dialogue, and tutor moves to increase efficiency, may be particularly helpful for high self-efficacy students. This group of students has more advanced self-regulated learning techniques and are receptive to feedback. For low self-efficacy students, guided experimentation may foster greater learning while at the same time potentially increasing frustration. This highlights the importance of carefully supporting low self-efficacy students during experimentation phases of computer science problem solving.

Future Work

As a research community, we must work to customize future learning environments to students' individual needs and preferences, with the goal of increasing both affective outcomes and learning. This study found that effective interactions between students of high and low self-efficacy vary, and the results suggest that it may be beneficial to have separate dialogue policies for the two groups. These policies could potentially increase the learning gain of students, while managing the frustration that they are experiencing while interacting with the challenging domain of Computer Science. In the future it will be important to take a deeper look into engagement and motivation, observing not only groups with higher likelihood of disengagement, but also the effectiveness of remedial strategies on different groups of students.

Another promising direction for future work is to investigate whether these findings are observed with younger learners. Computer science is being taught to students much younger than college age, and these students likely exhibit a much wider range of self-efficacy. Individual adaptation is a crucial goal for computer science tutoring systems, which hold great potential to support broader populations of students in learning computer science.

Acknowledgments This work is supported in part by the North Carolina State University Department of Computer Science and the National Science Foundation through Grant DRL-1007962, IIS-1409639, and CNS1453520. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the participants, and do not necessarily represent the official views, opinions, or policy of the National Science Foundation.

References

Anderson-Rowland, M. R., Bernstein, B. L., & Russo, N. F. (2007). The doctoral program in engineering and computer science: is it the same for women and men?. *Proceedings of the Women in Engineering Programs and Advocates Network 2007 Annual Conference*.

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychological Review, 84*(2), 191–215.

Bernacki, M., Nokes-Malach, T., & Aleven, V. (2015). Examining self-efficacy during learning: variability and relations to behavior, performance, and learning. *Metacognition and Learning, 10*(1), 99–117.

Bloom, B. (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.

Boyer, K. E., Vouk, M. A., & Lester, J. C. (2007). The influence of learner characteristics on task-oriented tutorial dialogue. *Proceedings of the 13th International Conference on Artificial Intelligence in Education (AIED),* 365–372.

Boyer, K. E., Phillips, R., Wallis, M., Vouk, M., & Lester, J. (2008). Balancing cognitive and motivationalscaffolding in tutorial dialogue. *Proceedings of the International Conference on Intelligent Tutoring Systems,* 239–249.

Boyer, K. E., Phillips, R., Ha, E. Y., Wallis, M. D., Vouk, M. A., Lester, J. C.et al. (2009). Modeling dialogue structure with adjacency pair analysis and hidden markov models. In *the Proceedings of the Conference of the North American Capture of the Association for Computational Linguistics and Human Language Technology (NAACL HLT)*, 49–52.

Chen, G., Gully, S. M., & Eden, D. (2001). Validation of a new general self-efficacy scale. *Organizational Research Methods, 4*(1), 62–83.

Chen, L., Di Eugenio, B., Fossati, D., Ohlsson, S. & Cosejo, D. (2011). Exploring effective dialogue act sequences in one-on-one computer science tutoring dialogues. *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications*, 65–75.

Chi, M. T. H., Siler, S. A., Jeong, H., Yamauchi, T., & Hausmann, R. G. (2001). Learning from human tutoring. *Cognitive Science, 25*(4), 471–533.

Chi. M., VanLehn, K. & Litman, D. (2010). Do micro-level tutorial decisions matter: applying reinforcement learning to induce pedagogical tutorial tactics. *Proceedings of the International Conference on Intelligent Tutoring Systems*, 224–234.

Croy, M., Barnes, T., & Stamper, J. (2007). Towards an intelligent tutoring system for propositional proof construction. In Brey, P., Briggie, A., Waelbers, K. (eds.) *European computing and philosophy conference*, 145–155.

D'Mello, S. K., Williams, C., Hays, P., & Olney, A. (2009). Individual differences as predictors of learning and engagement. *Proceedings of the Annual Meeting of the Cognitive Science*, 308–313.

D'Mello, S. K., Lehmnan, B., Pekrun, R., & Graessar, A. (2014). Confusion can be beneficial for learning. *Learning and Instruction, 29*, 153–170.

DiSalvo, B., Yardi, S., Guzdial, M., McKlin, T., Meadows, C., Perry, K., et al. (2011). African American men constructing computing identity. *Proceedings of the 2011 annual conference on Human factors in computing systems - CHI*, 2967–2970.

du Boulay, B., Avramides, K., Luckin, R., Martinez-Miron, E., Rebolledo-Mendez, G., & Carr, A. (2010). Towards systems that care: a conceptual framework based on motivation, metacognition and affect. *International Journal of Artificial Intelligence in Education, 20*(3), 197–229.

Dweck, C.S. (2002). The development of ability conceptions. In A. Wigfield & J.S. Eccles (Eds.), *Development of achievement motivation: a volume in the educational psychology series*, 57–88. Academic Press.

Dzikovska, M., Steinhauser, N., Moore, J.D., Campbell, G.E., Harrison, K.M. & Taylor, L.S. et al. (2010). Content, social, and metacognitive statements: an empirical study comparing human-human and human-computer tutorial dialogue. *Proceedings of the European Conference on Technology Enhanced Learning*, 93–108.

Ezen-Can, A., & Boyer, K. E. (2015). Understanding student language: an unsupervised dialogue act classification approach. *International Journal of Educational Data Mining (JEDM), 7*(1), 51–78.

Forbes-Riley, K., & Litman, D. (2005). Using bigrams to identify relationships between student certainness states and tutor responses in a spoken dialogue corpus. *Proceedings of the 6th SIGdial Workshop on Discourse and Dialogue*.

Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C. W., Chen, L., & Cosejo, D. G. et al. (2009). I learn from you, you learn from me: How to make iList learn from students. *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED)*, 491–498.

Gerdes, A., Jeuring, J., & Heeren, B. (2012). An interactive functional programming tutor. *Proceedings of the 17th ACM annual congerence on Innovation and technology in computer science education*, 250–255.

Graesser, A. C., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R., & Tutoring Research Group. (1999). AutoTutor: a simulation of a human tutor. *Cognitive Systems Research, 1*(1), 35–51.

Grafsgaard, J., Wiggins, J., Boyer, K., Wiebe, E., & Lester, J. (2013). Embodied affect in tutorial dialogue: student gesture and posture. *Proceedings of the 16th International Conference on Artificial Intelligence in Education*, 1–10.

Grafsgaard, J., Lee, S., Mott, B., Boyer, K., & Lester, J. (2015) Modeling self-efficacy across age groups with automatically tracked facial expressions. *Proceedings of the International Conference on Artificial Intelligence in Education*, 582–585.

Hardy, M., Wiebe, E., Grafsgaard, J., Boyer, K. E., & Lester, J. (2013). Physiological responses to events during training: use of skin conductance to inform future adaptive learning systems. *Proceedings of the International Conference on Affective Computing and Intelligent Interaction (ACII)*, 2101–2105.

Hart, S.G. & Staveland, L.E. (1988). Development of NASA-TLX (Task Load Index): Results of empirical and theorectical research. human mental workload. P.A. Hancock and N. Meshkati, eds. Elsevier Science, 139–183.

Howley, I., Adamson, D., Dyke, G., Mayfield, E., Beuth, J., & Rose, C. et al. (2012). Group composition and intelligent dialogue tutors for impacting students' academic self-efficacy. *Intelligent Tutoring Systems*, 551–556.

Koch, M. (1994). No girls allowed! *TECHNOS, 3*(3), 14–19.

Lane, H. C., & VanLehn, K. (2005). Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education, 15*, 474–479.

Lane, H. C., Cahill, C., Foutz, S., Auerbach, D., Noren, D., Lussenhop, C., et al. (2013). The effects of a pedagogical agent for informal science education on learner behaviors and self-efficacy. In *Artificial Intelligence in Education*, 309–318.

Lepper, M.R., Woolverton, M., Mumme, D.L. & Gurtner, J.L. (1993). Motivational techniques of expert human tutors: lessons for the design of computer-based tutors. Computers as Cognitive Tools, 75–105.

Litman, D., Moore, J.D., Dzikovska, M. & Farrow, E. (2009). Using natural language processing to analyze tutorial dialogue corpora across domains and modalities. *Proceedings of the 14th International Conference on Artificial Intelligence in Education*, 149–156.

Marx, J. D., & Cummings, K. (2007). Normalized change. *American Journal of Physics, 75*(1), 87–91.

McKendree, J. (1990). Effective feedback content for tutoring complex skills. *Human-Computer Interaction, 5*(4), 381–413.

Mitchell, C., Ha, E., Boyer, K., & Lester, J. (2013). Learner characteristics and dialogue: recognizing effective and student-adaptive tutorial strategies. *International Journal of Learning Technology (IJLT), 8*(4), 382–403.

Narciss, S. (2013). Designing and evaluating tutoring feedback strategies for digital learning environments on the basis of the interactive tutoring feedback model. *Digital Education Review, 23*, 7–26.

Pekrun, R., Goetz, T., Daniels, L., Stupnisky, R. H., & Perry, R. P. (2010). Boredom in achievement settings: exploring control-value antecedents and performance outcomes of a neglected emotion. *Journal of Educational Psychology, 102*(3), 531–549.

Rivers, K., & Koedinger, K. (2012). A canonicalizing model for building programming tutors. *Proceedings of the International Conference on Intelligent tutoring systems*, 591–593.

Sadker, D. (1999). Gender equity: still knocking at the classroom door. *Educational Leadership, 56*, 22–27.

Symonds, M. R. E., & Moussalli, A. (2010). A brief guide to model selection, multimodel inference and model averaging in behavioral ecology using Akaike's information criterion. *Behavioral Ecology and Sociobiology, 65*(1), 13–21.

Vail, A. K., Boyer, K. E., Wiebe, E. N., & Lester, J. C. (2015). The Mars and Venus effect: the influence of user gender on the effectiveness of adaptive task support. *The 23rd Conference on User Modelling, Adaptation and Personalization*, 265–276.

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist, 46*(4), 197–221.

VanLehn, K., Graesser, A. C., Jackson, G. T., Jordan, P., Olney, A., & Rose, C. P. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science, 31*(1), 3–62.

Wang, S.-L., & Wu, P.-Y. (2008). The role of feedback and self-efficacy on Web-based learning: the social cognitive perspective. *Computers & Education, 51*, 1589–1598.

Wiebe, E. N., Williams, L., Yang, K., & Carol Miller, C. (2003). *Computer science attitude survey. (Report No.: NCSU CSC TR-2003-1)*. Raleigh: Dept. of Computer Science, NC State University.

Xu, S., & Chee, Y. S. (2003). Transformation-based diagnosis of student programs for programming tutoring systems. *Software Engineering, IEEE Transactions on, 29*(4), 360–384.