Computer Science Education, vol. 19, iss. 2, pp. 111-136
Author's preprint version.

Investigating the Role of Student Motivation in Computer Science Education through One-on-One Tutoring

Kristy Elizabeth Boyer,[§*] Robert Phillips,[§+] Michael D. Wallis,[§+]
Mladen A. Vouk,[§] and James C. Lester[§]

[§] Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA

[+] Applied Research Associates, Inc.
Raleigh, North Carolina, USA

[*] Corresponding author: keboyer@ncsu.edu

Investigating the Role of Student Motivation in Computer Science Education through One-on-One Tutoring

ABSTRACT. The majority of computer science education research to date has focused on purely cognitive student outcomes. Understanding the *motivational* states experienced by students may enhance our understanding of the computer science learning process, and may reveal important instructional interventions that could benefit student engagement and retention. This article investigates issues of student motivation as they arise during one-on-one human tutoring in introductory computer science. The findings suggest that the choices made during instructional discourse are associated with cognitive and motivational outcomes, and that particular strategies can be leveraged based on an understanding of the student motivational state.

1. Introduction

Research in computer science education explores how students develop an understanding of computation and computational systems. Such research is informed by an emerging view of learning as a complex process involving cognitive, affective, and motivational states. The *cognitive* aspect of learning involves memory and reasoning about the subject matter, while *affective* processes pertain to the emotional states experienced by the student. *Motivation* refers to a student's impetus for engaging in learning activities. Together these dimensions provide a model that can be used to explore how students come to understand computer science.

Research into cognitive learning issues informs the design of instructional interventions that facilitate the construction of computing knowledge. Similarly, an understanding of the

motivational states experienced by students may inform a variety of approaches that enhance the learning process. For example, a student who lacks motivation due to low self-confidence may be spurred to continue working with an educational software system when the system provides positive feedback (Tan & Biswas, 2006). Conversely, certain instructional tactics such as explicit reassurance are best avoided with students who are already in a state of high motivation (*e.g.*, Rebolledo-Mendez, Boulay, & Luckin, 2006).

This article presents results obtained from examining student motivation during one-on-one human tutoring sessions involving experienced human tutors and novice computer science students. Tutoring has been recognized as an environment in which effective teaching strategies for computer science can be applied (Ragonis & Hazzan, 2008), and one-on-one tutoring is an excellent testbed for exploring research questions in computer science pedagogy because the constrained interaction facilitates *control* of conditions, *capture* of the instructional interaction, and *coverage* of different teaching scenarios beyond what is easily feasible in whole-class research studies (Boyer, Phillips, Wallis, Vouk, & Lester, 2009; Cade, Copeland, Person, & D'Mello, 2008).

This article makes three contributions. First, it presents the argument for studying student motivation as it occurs over a single period of learning. Second, it makes the case that one-on-one human tutoring is a suitable testbed for exploring cognitive phenomena in computer science, and moreover, that human tutoring is an informative means by which student motivation and affect can be carefully investigated. Finally, it presents the *corpus analysis* research methodology and the resulting empirical findings from three separate exploratory studies involving human tutors and novice computer science students.

The results to date suggest that the motivational components of *control* and *confidence* (Lepper, Woolverton, Mumme, & Gurtner, 1993) may influence computer science learning interactions in important ways, and that these motivational components may themselves be influenced to the student's benefit through specific instructional strategies.

This article is structured as follows. Background on the role of motivation and tutorial dialogue from related literature is discussed in Section 2. Section 3 presents the design of three exploratory research studies and describes the structure of the resulting data from human-human tutoring of introductory computer science. Section 4 describes the research methodology applied to the data sets of tutorial dialogue. Section 5 presents the results with subsections devoted to the motivational components of *confidence* (Sections 5.1 and 5.2) and *control* (Section 5.3). Discussion follows in Section 6, and conclusions are presented in Section 7.

2. Background

The overwhelming majority of computer science education literature has focused on the purely cognitive aspect of learning (Machanick, 2007). This trend is not surprising given the alluring parallels between cognitive learning models and the basic functions of computing that are fundamental to the discipline. For instance, the theoretical framework known as *constructivism* has been embraced for its insights into CS learning processes (Ben-Ari, 1998), and direct analogies are sometimes made between the constructivist view, in which students build and "debug" knowledge, and the activities involved in computer programming. Constructivism and other purely cognitive models of learning (*e.g.*, Bloom, 1956) are valuable in understanding many phenomena surrounding the teaching and learning of computing. However, these models may not capture some important facets of the computer science learning process.

As Machanick (2007) observes, there exist phenomena in CS education that are not readily explained by current purely cognitive frameworks. He proposes that *social constructivism,* a theoretical framework that is gaining acceptance in the broader education community, might offer explanations for the observed effectiveness of some approaches such as peer assessment and apprenticeship-style teaching (Guzdial & Tew, 2006). The potential insights afforded by social constructivism stem from the theory's foundational tenets that learning has important social roles, and that communication is key to defining the knowledge of a student. Evidence of the importance of communication in computer science learning environments has been noted by Barker and Garvin-Doxas (2004), who observe that the type of discourse that occurs in a computing classroom has far-reaching effects on students. Further results on the importance of communication and the social role of learning have emerged from research in the context of pair programming (Slaten, Droujkova, Berenson, Williams, & Layman, 2005) and non-majors learning to program (Wiedenbeck, 2005).

One instructional setting that has been proven effective in building knowledge and is rich in communication between student and instructor is one-on-one human tutoring. Long studied as an exemplary way to facilitate mastery of a subject (*e.g.*, Bloom, 1984), tutoring has been the setting for recent work in CS education research, for example, in investigating how students plan the solution to a programming problem (Lane & VanLehn, 2005). When the conversation between tutor and student is captured, a corpus of *tutorial dialogue* is created that constitutes a record of the tutorial interaction. Tutorial dialogue has been studied in depth by researchers from disciplines including cognitive psychology and artificial intelligence. Foundational results in this area have revealed, for example, that tutorial dialogues often exhibit a regular, iterative structure that unfolds as tutor and student work through problems together (Graesser, Person, & Magliano,

1995). In addition, extensive work has shown the importance of interaction as a source of effectiveness for one-on-one tutoring (Chi, Siler, Jeong, Yamauchi, & Hausmann, 2001). Finally, recent results suggest the effectiveness of tutoring sessions depends on the preparedness of the students at the outset (VanLehn et al., 2007).

Because of the completeness of the instructional record created by controlled tutorial dialogue studies, it is possible to observe and make inferences about the fine details of student activities. Patterns observed in the exploratory studies presented here indicate that *student motivation* is an important facet of computer science learning. The findings suggest several hypotheses regarding the ways in which particular approaches may be implemented in practice to increase the motivational effectiveness of instructional dialogue in computer science.

Motivation, which refers to a student's impetus for engaging in learning activities, has received attention in the general education research community for at least two decades (*e.g.*, Cameron & Pierce, 1994; Deci, Koestner, & Ryan, 2001; Keller, 1983). Recently, motivating the student has also been identified as a component of a complete conception of teaching computer science (*e.g.*, Lister et al., 2007). Student motivation has also been considered in several recent empirical studies in computer science education. For example, Soh, Samal, & Nugent (2007) included attitudinal variables for student self-efficacy and motivation as part of a data collection effort to assess the effectiveness of a redesigned computer science curriculum. Additionally, pair programming researchers recognize motivation as an important facet when measuring the impact of pair programming in educational settings (*e.g.*, Williams, Wiebe, Yang, Ferzli, & Miller, 2002). These studies show an increased awareness of the importance of motivation in the computer science learning process.

In much of the existing computer science education research, measures of motivation are taken at the beginning and end of an academic term in order to assess the impact of the instructional approach utilized during the term. Tracking changes at this granularity has proven a useful research approach. However, studying student motivation at a finer granularity, for instance, over the course of a single programming assignment, can complement the coarser granularity approach generally undertaken to date. For example, Wolfe (2004) considers student motivation at the level of a single programming assignment, observing that the rhetoric used in problem descriptions does matter. Wolfe's concrete finding is that programming assignments may be more successful if they emphasize real-world purpose and human factors. This kind of contribution is made possible by studying student motivation at a finer granularity than over entire academic terms.

The study of human tutoring provides a means through which student motivation can be examined closely. In this setting, instructional strategies for motivation can be examined as they naturally occur, and the impact of specific approaches can be measured at the level of individual students. Lepper et al. (1993) studied the tactics of expert human tutors in various academic domains to inform a theory of how teachers take motivation into account while instructing students. They posit that four dimensions of motivation can be influenced to enhance instruction: *control*, *curiosity*, *challenge*, and *confidence*. This work provides a theoretical framework for investigating human-human tutoring and emphasizes the importance of fine-grained motivational analysis even down to the dialogue turn level.

This article reports on three exploratory tutorial dialogue studies we conducted to investigate relationships between student learning, motivation, and specific dialogue phenomena that occur during tutoring. Results speak to the motivational components of *control* and

confidence as they manifest during tutoring sessions centered around the task of solving an introductory programming exercise. Each study involved naturalistic interaction between introductory computer science students and human tutors of varying experience levels. The instructional interaction took place over the course of approximately one hour while students worked to solve a programming exercise. The studies were conducted in a controlled setting so that all interaction between student and tutor could be logged for analysis. The exploratory findings suggest that student motivation is an important component of the processes by which students come to understand computing.

3. Design of Exploratory Studies

This section describes the three tutoring studies. Study I was conducted in Fall semester 2006, Study II was conducted in Spring semester 2007, and Study III was conducted in Spring semester 2008.

3.1 Participants

Students were volunteers who were enrolled in an introductory university-level computer science course called "Introduction to Computing – Java" at the time of the study. This course is the first in a series of computing courses offered at the university. Study I involved 35 participants, Study II involved 43, and Study III included 61 participants. Students were compensated for participation through a small amount of class credit that varied according to instructor preference. Almost all students were of traditional college age, and, as in the larger class populations, the majority of participants were male. Not all participants planned to major in computer science; enrollment in the introductory computer science course, and therefore participation in the studies reported here, included students whose declared majors were

mechanical, electrical, and computer engineering. The studies began in the eighth week of the semester. Studies I and II each spanned one week, and Study III spanned two weeks.

3.2 Tutors

Study I utilized six tutors: four graduate students, one female and three male, and two upper-division undergraduate students, both male. Study II used fourteen tutors: twelve graduate students, two female and ten male, plus two upper-division male undergraduate students. Study III involved the two most effective tutors from the prior studies, that is, the tutors who had the highest average student learning gains across Study I and Study II. The tutors in Study III were one female graduate student and one male upper-division undergraduate student. All tutors across the studies were between the ages of 19 and 30. All tutors had a minimum of one semester's experience as a tutor or teaching assistant. Several tutors in Study II also had extensive experience as classroom instructors. None of the tutors were involved as instructors or teaching assistants with the course from which the participants were drawn. Neither students' nor tutors' identities were revealed before, during, or after the tutoring sessions.

The tutor orientation consisted of a problem-solving session in which all of the tutors met to work through alternate solutions to the exercise in order to refresh tutor knowledge and establish a breadth of solutions to the programming problem. In addition, tutors were shown the student instruction video in order to familiarize them with the assumed starting knowledge of the student regarding the software being used. The student instruction video also served as the tutor orientation to the software. Tutors were not instructed to use specific instructional approaches or tutorial strategies because the intent was for each tutor to use his or her own strategies to accomplish the goal of helping students complete a programming exercise while ensuring that students developed an understanding of the general concepts used in the solution. In this way,

the data represent a sampling of naturalistic human-human tutoring for introductory computer science.

3.3 Problem-Solving Task

The tutorial dialogue was centered around a problem-solving task. For this task, Studies I and II used a programming exercise taken from the standard laboratory manual for the course (Appendix A). In these studies, students attended the tutorial session in lieu of attending their regularly scheduled structured lab for the week in order to ensure that students had not already completed the programming exercise when they arrived to work with the tutor. The programming exercise focused on the use of array data structures and loop constructs. Students were provided a partial solution that included an (initially empty) graphical display of the generated results. Students were required to complete three code modules to solve the programming problem. Studies I and II were time-controlled at 50 and 55 minutes, respectively. Most students completed two of the three methods during the allotted tutoring time. Based on tutor feedback from the previous two studies which indicated the programming problem was unnecessarily confusing for students, Study III used a slightly simplified programming exercise (Appendix B) that was designed with social relevance in mind, a property thought to be implicitly motivational to students (Layman, Williams, & Slaten, 2007). As in the previous studies, the programming exercise focused on using array data structures and loop constructs to complete three modules. In Study III, rather than controlling for time, students were permitted to work until completion of the programming exercise.

3.4 Procedure

Upon arrival, participants completed a pair of written instruments consisting of survey items on the student's motivation to study computer science, including the student's confidence (Appendix C). The confidence components of these motivation surveys were adapted from the Domain-Specific Self-Efficacy Scale (Bandura, 2006).

In Study I, this data was the first to be collected for each participant; in Studies II and III, participants were also asked to complete an electronic survey containing several demographic and psychometric instruments prior to arriving for the study. The demographic instrument collected students' ethnicity, expected graduation date, and major. Psychometric instruments included the Achievement Goals Questionnaire (Elliot & McGregor, 2001) and the Interpersonal Reactivity Index (Davis, 1983). These instruments were not analyzed for the results reported here.

The pretests and posttests (Appendix D) were handcrafted and evolved between studies in an effort to make the questions more sensitive to differences in learning that occurred over the course of the tutoring sessions. The tests for Studies I and II underwent no external evaluation; for Study III, the pre/post test underwent formal review by a panel of three independent subject matter experts with experience in teaching introductory computer science.

Upon completing the written instruments, students were seated at a computer where they watched a short instructional video to familiarize them with the software that would be used. Upon completion of the instructional video, students worked remotely with a tutor while planning and implementing the solution to the programming exercise. When the tutoring session reached its conclusion, a paper-based post-survey and posttest were administered whose items were analogous to the pre-survey and pretest.

3.5 Tutorial Interaction

In order to fully capture the interaction between student and tutor, students reported to a separate room from their tutor and worked remotely with the tutor using software designed to facilitate real time remote collaboration on programming projects (Boyer, Dwight, Fondren, Vouk, & Lester, 2008). Using this software package, the student and tutor engaged in dialogue through a textual interface similar to mainstream chat messaging programs (Figure 1). As students constructed Java code in the Eclipse IDE (The Eclipse Foundation, 2009), an Eclipse plug-in transmitted the student's problem-solving actions to the tutor in real time. The tutor was thus able to observe student problem-solving actions (*e.g.*, programming, scrolling, executing) continuously throughout the tutoring session. Tutors were limited to viewing students' programs; only students could edit programs. All interactions between the student and tutor, as well as all student programming actions, were logged to a database for further analysis.

Tutors and students were not aware of each other's identity. No individual characteristics (including gender, ethnicity, age, or level of preparedness) were disclosed to the tutor or the student. This restriction was communicated to all participants ahead of time. In the rare event that students inquired as to the tutor's identity, tutors were instructed to redirect the student with a response such as, "Sorry, we're supposed to talk only about the programming exercise." The need for this redirection arose infrequently in the studies, but was necessary to ensure that student and tutor assumptions would be controlled to the fullest extent possible.

In Study I, there were no restrictions placed on the construction of dialogue messages; that is, while one user actively constructed a textual message, the other user was also permitted to construct and send messages. This design choice was made because of its consistency with the interface design of commercial instant messaging platforms familiar to the student

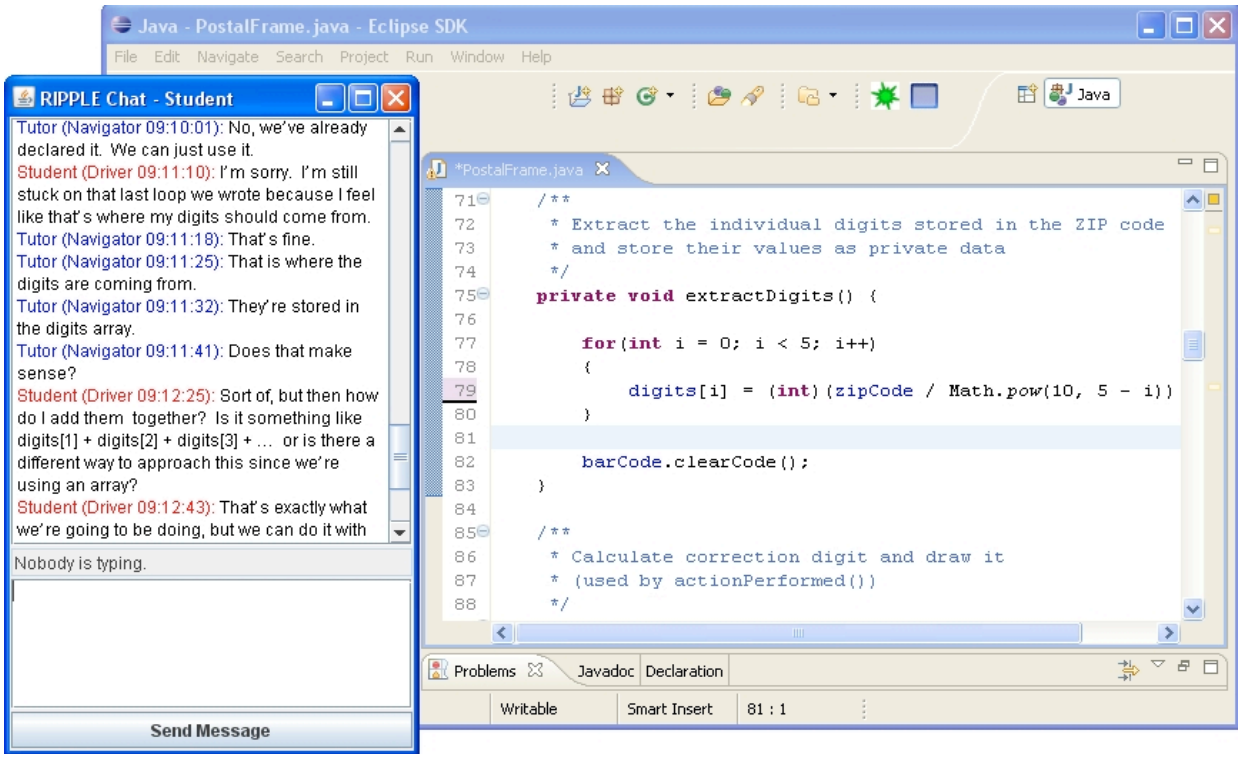


Figure 1: Tutorial dialogue and problem-solving interface

population. In these instant messaging platforms, if one user completes a new message (possibly starting a new topic) while the other user is typing a response to the previous topic, the chronological record of dialogue can appear inconsistent with respect to the conversational structure. Human users deal with this phenomenon readily as the textual dialogue unfolds in real time; however, the situation gives rise to analysis challenges because researchers must "untangle" the logs manually before analysis. To address this issue, the dialogue interface was modified for Studies II and III to enforce strict turn-taking. When a user was actively constructing an utterance in the textual dialogue interface, the other user was not permitted to construct an utterance. However, the student was permitted to continue working in the problem-solving window regardless of the status of the textual dialogue interface.

4. Methodology

Corpus analysis is a research methodology commonly used in computational linguistics (Jurafsky & Martin, 2008). The approach involves first obtaining a body of text called a *corpus* and then marking this text for features that are not explicitly present on its surface in a process called *annotation.* The marks that are applied are often referred to as *tags.* In an effort to describe the corpus in a concise way or to establish relationships between qualities in the corpus and other external entities of interest, the occurrences of the tags are often modeled using quantitative methods. This section presents the results of applying this general corpus analysis methodology involving two different levels of annotation: *dialogue act* annotation, which involves marking each utterance with its intended purpose (*e.g.*, question, feedback), and *initiative* annotation, which involves marking larger sections of the dialogue to indicate which of the participants was leading the dialogue at each point (*e.g.*, tutor or student).

4.1 Data

The data traces generated by these studies include student programming actions at the keystroke level, as well as all textual dialogue utterances between participants. Each of the three studies yielded a chronological log with over five thousand dialogue moves and tens of thousands of student programming actions. These data constitute a task-oriented corpus of *tutorial dialogue* to which widely-used corpus analysis techniques, beginning with corpus annotation, were applied. Through the process of *annotation,* the raw corpora were transformed into meaningful representations of the tutoring interactions. The remainder of this section describes the annotation schemes.

4.2 Dialogue Act Annotation

Employed in Studies I and II, dialogue act annotation involves marking each dialogue move with a tag summarizing the utterance's purpose (*e.g.*, greeting, questioning, answering, disagreeing). For example, in tutorial dialogue, examples of common dialogue acts include asking questions (*e.g.*, "What kind of variable should I use?"), making assessments of knowledge (*e.g.*, "I don't know how to declare an array."), and acknowledging a previous statement (*e.g.*, "Got it."). Because there is no gold standard for annotating tutorial dialogue, our set of dialogue act tags was adapted from annotation schemes from the dialogue analysis literature to capture the salient characteristics of the corpora. Some dialogue acts were taken directly from a set applied in the domain of qualitative physics (Forbes-Riley, Litman, Heuttner, & Ward, 2005), while other tags were inspired by a more expansive set of tags created for general natural language dialogue (Stolcke et al., 2000). Because a single utterance might communicate cognitive, affective, and motivational content, the set of dialogue act tags was divided into two channels: an affective/motivational channel and a cognitive channel.[1] Table 1 illustrates the dialogue act tags as applied during Study II; Study I involved further distinctions within the *Question* tag and the *Feedback* tags, but these distinctions were not made for Study II.

4.3 Problem-Solving Act Annotation

The corpora contain a variety of problem-solving actions taken by the student. These programming actions include opening and closing files, typing new text in the program editor window, and scrolling through program files. Most of these categories of programming actions were sparse and were therefore eliminated from analyses, leaving the two most common

[1] The tagging scheme was divided into cognitive and affective/motivational channels. Although the analysis presented here focuses only on the motivational tags in the affective/motivational channel, the entire tagging scheme is presented for completeness.

Table 1: Dialogue Act Tagging Scheme for Study II

	Act	Description	Tutor and Student Example Utterances	Average Count Per Session (Standard Deviation) Student	Tutor
Cognitive Channel	Question (Q)	Questions about goals to pursue, domain concepts, etc.	"Where should we start?" "How do I declare an array?"	6.5 (4.2)	0.6 (0.8)
	Evaluative Question (EQ)	Questions that explicitly inquire about student knowledge state or correctness of problem-solving action.	"Do you know how to declare an array?" "Is that right?"	9.7 (7.1)	7.0 (5.0)
	Statement (S)	Declarative assertion.	"You need a closing bracket there." "I am looking for where this method is declared.."	4.9 (4.1)	46.2 (21.0)
	Acknowledgement (ACK)	Positive acknowledgement of a previous statement.	"Okay." or "Yeah." "Alright."	3.8 (5.0)	2.5 (1.9)
	Extra Domain (EX)	A statement not related to the computer science discussion.	"Hello" or "You're Welcome" "Can I use my book?"	1.0 (2.1)	1.1 (2.4)
	Positive Feedback (PF)	Unmitigated positive feedback regarding problem solving action or student knowledge state.	"Yes, I know how to declare an array." "That is right."	2.7 (2.5)	12.0 (7.6)
	Lukewarm Feedback (LF)	Partly positive, partly negative feedback regarding student problem solving action or student knowledge state.	"Sort of." "You're close." or "Well, almost."	0.7 (1.2)	2.3 (2.5)
	Negative Feedback (NF)	Negative feedback regarding student problem solving action or student knowledge state.	"No." "Actually, that won't work."	2.1 (1.8)	1.3 (2.1)
Motivational/Affective Channel	Confusion (C)	Explicit expression of confusion. Indicates disorientation beyond that indicated by negative feedback (which indicates the student lacks a particular piece of knowledge).	"I have no idea what to do." "I'm lost."	0.8 (1.2)	-
	Frustration (F)	Explicit expression of frustration.	"Grrr!" "This is so frustrating."	0.1 (0.4)	0.0 (0.3)
	Excitement (E)	Explicit expression of excitement.	"Sweet!" "Cool!"	0.4 (0.7)	0.3 (0.6)
	Praise (P)	Statement intended to emphasize a student's success. This goes beyond positive feedback, which serves as factual confirmation of correctness.	"Great job on that part!" "That's perfect."	-	4.2 (5.7)
	Reassurance (R)	Statement intended to minimize a student's failure by providing comfort.	"That part was hard." "Don't worry about it."	0.4 (0.6)	1.3 (1.6)
	Other Emotion (O)	Utterance that conveys affective or motivational content but for which there is no pre-defined tag.	"Ha ha." "I'm sorry."	0.7 (1.0)	1.5 (2.1)

programming actions of *typing* Java code in the programming interface and *scrolling* in the program editor. The events were automatically tagged using a heuristic for correctness: if a problem-solving action was a programming keystroke that survived until the end of the session, this event was tagged *promising*, to indicate it was probably correct; if a problem-solving act was a programming keystroke that did not survive until the end of the session, the problem-solving act was tagged *questionable*. Both of these heuristics are based on the observation that in this tutoring context, students solved the problem in a linear fashion and tutors did not allow students to proceed past a step that had incorrect code in place. Finally, periods of consecutive scrolling were also marked *questionable* because in a problem whose entire solution fits on one printed page, scrolling was usually conducted in irrelevant source files included to support graphical output of the programming exercise. Because the student's solution did not interface directly with these source files, scrolling through them was almost uniformly not a productive problem-solving step. This automatic tagging for correctness was applied for the purposes of Study II.

4.4 Annotation for Initiative

While dialogue act annotation involves marking a corpus at the level of dialogue *turns*, another useful type of annotation entails marking the higher-level structure of the dialogue. Tags at the dialogue structure level can span many individual dialogue acts. Since the corpora consist of dialogue turns interleaved chronologically with student problem-solving actions, dialogue structure tags span contiguous sections of textual dialogue and student problem-solving. One important aspect of dialogue structure involves *initiative*, that is, which speaker has control and direction of the conversation at a given point in time. This level of annotation was performed in Study III. The tutorial dialogue corpora lent themselves readily to two tags for initiative: Student-Initiative and Tutor-Initiative.

In *Student-Initiativ*e mode, the student maintains control and direction over the problem-solving effort. Student-Initiative mode is characterized by the following activities:

- The student states his/her plan and (optionally) asks the tutor for feedback.
- The student reads the problem description or constructs a portion of the actual solution independently, as indicated by no dialogue exchanged while the student is conducting these problem-solving activities.
- The student asks content-based questions (*e.g.*, "I should start this index at 0, right?") as opposed to content-free questions (*e.g.*, "What do I do now?").

In *Tutor-Initiative* mode, the tutor directs the problem-solving effort. Because the user interface does not allow tutors to edit the students' solutions, Tutor-Initiative mode does not involve the tutor actively constructing the problem solution. However, the tutor often used the textual dialogue interface to actively guide and direct the student to take very specific problem-solving actions. Tutor-Initiative mode includes the following activities:

- The tutor offers unsolicited advice or correction.
- The tutor lectures on a concept.
- The tutor explicitly suggests the next step in problem solving.
- The tutor poses questions to the student.

To illustrate the initiative modes, two excerpts are presented (Table 2). The first excerpt illustrates Student-Initiative mode. In this excerpt, the student asks a content-based question indicating he knows the problem lies in a return statement. The tutor provides an answer, which the student acknowledges. Finally, the student spends five uninterrupted minutes coding part of the problem solution. Lengthy periods of independent student work are common in Student-Initiative mode. The second excerpt illustrates Tutor-Initiative mode. In this excerpt, the tutor

Table 2. Excerpts of Student-Initiative and Tutor-Initiative modes from Study III

Student-Initiative Mode

Student:	What am I not typing right in the return statement?
Tutor:	You only need to return the identifier.
Tutor:	In other words, you just need to return newtimes.
Student:	Ok.
	[student works independently for five minutes]

Tutor-Initiative Mode

Tutor:	Hmm, that doesn't look quite right.
Tutor:	Do you see the projected array output?
Student:	Yes.
Tutor:	It looks like it's only getting the first value…
Tutor:	So your loop must be stopping before it's done with its work.
Tutor:	Do you see what might be causing that?
	[tutor-led conversation continues]
Tutor:	But it's coming out 1.0 instead of 4.3.
Tutor:	Anything else look wrong on the graph, compared with the instructions?
Student:	The second bar is not right.
Tutor:	I think fixing the length might be the only thing you need to change.
	[student works for ten seconds]
Tutor:	Much better.
Student:	Yeah!!

gives unsolicited advice and asks questions of the student. The student spends a brief time repairing the problem solution, and the tutor once more provides unsolicited feedback. As illustrated in this excerpt, brief periods of student work interspersed with frequent dialogue are common in Tutor-Initiative mode.

5. Results

This section presents results from three tutorial dialogue studies that employ a mixed methods research methodology: data sources include both quantitative data gained from survey responses

and test scores (Section 3.4), along with qualitative data resulting from the interpretation (through tagging) of textual dialogues. A mixed methods approach is appropriate when the research questions involve *why* certain quantitative results have been observed (Creswell & Clark, 2006). In the current work, tutorial dialogue structure is hypothesized to explain, at least in part, quantitative results involving learning and confidence.

The first result deals with the ways in which human tutors naturally adapt to student characteristics such as self-confidence. The second set of findings suggest that specific types of tutorial feedback may be associated with different student confidence outcomes, while the third result deals with learning and confidence as they relate to the level of initiative taken by the tutor during the tutoring sessions. Together, these exploratory results inform hypotheses regarding the role of student motivation, and how instructional strategies might address it, in computer science learning.

5.1 Study I: Tutorial Adaptation to Student Confidence

Self-confidence is an important component of student motivation. The results from Study I suggest that tutors may adapt their strategies in specific ways based on student confidence. Although no student characteristics were explicitly revealed to the tutors, the dialogue structure of students with low self-confidence differed significantly from that of students with high self-confidence.

The corpus from Study I consists of 5,034 dialogue acts: 3,075 tutor turns and 1,959 student turns. The entire corpus was manually annotated for dialogue acts by a single researcher, with a second researcher annotating a subset of 969 total utterances. An agreement study to evaluate the consistency of the coding scheme and its application to the corpus found a Kappa agreement statistic of 0.75, indicating reasonable inter-rater reliability (Cohen, 1960). In order to

compare tutorial sessions, the relative frequency of each dialogue act was computed as the ratio of the number of occurrences of that dialogue act to the total number of dialogue acts in the session (Boyer, Vouk, & Lester, 2007).

Overall, the tutoring sessions in Study I were effective: on average, students scored 13% higher on the posttest than the pretest. This average learning gain is statistically significant ($p<0.0001$ using a *t*-test with 34 *DF, SD*=0.12*)* and the effect size is 1.08. For further analysis, students were divided into two groups based on the self-reported confidence score. This measure was obtained from a pre-survey item in which students were asked to rate their own confidence, on a scale of 0-100, that they could complete a simple programming exercise on their own. Students were classified as being in one of two groups, *highly-confident* or *less-confident,* according to whether the self-reported confidence level of that participant fell above or below the median reported confidence level of all participants.

Some dialogue acts occurred with significantly different relative frequencies between the two confidence groups. The *relative frequency* of the following dialogue acts was significantly different between the highly-confident and less-confident groups:

- Students in the highly-confident group made more declarative statements, or assertions, than students in the less-confident group ($p=0.044$, *t*-test with unequal variances, 22.8 *DF*, SD_{high}=0.04, SD_{low}=0.01).
- Tutors paired with less-confident students gave more negative feedback ($p=0.021$, *t*-test with unequal variances, 15.7 *DF*, SD_{high}=0.0009, SD_{low}=0.0054) and made fewer acknowledgements ($p=0.05$, *t*-test with unequal variances, 21.9 *DF*, SD_{high}=0.03, SD_{low}=0.008) than tutors paired with highly-confident students.

The relative frequencies of other dialogue acts showed no significant difference between the groups. These findings suggest the hypothesis that differing levels of student self-confidence are associated with structural differences in the tutor-student interaction. This result highlights the importance of further study of these phenomena.

5.2 Study II: Impact of Corrective Feedback on Student Motivation

Although the learning gain results indicate that the tutoring sessions were effective, Study I did not indicate which tutorial adaptations might be more or less effective from either a cognitive or a motivational perspective. To address this limitation, Study II (Boyer, Phillips, Wallis, Vouk, & Lester, 2008) examined the impact of certain cognitive and motivational corrective strategies focusing on three categories of dialogue acts utilized by tutors. The motivational strategies of *praise* and *reassurance* were compared with several types of *cognitive feedback* to identify relationships with student cognitive and motivational outcomes. In order to focus the analysis, these strategies were analyzed when they were used immediately following plausibly incorrect, or *questionable*, student problem-solving actions.

The corpus from Study II consists of 4,864 dialogue moves: 1,528 student utterances and 3,336 tutor utterances. All dialogue moves were annotated for dialogue acts using the dialogue act tag set shown in Table 1. As in Study I, an agreement study was conducted. A second researcher annotated 1,418 utterances from the corpus, and the resulting Kappa agreement statistic was 0.76, indicating reasonable inter-rater reliability. In addition to dialogue act tagging, all student programming actions were automatically annotated using the problem-solving annotation described earlier. Of the 3,336 tutor utterances, 1,243 occurred directly after a student problem-solving action that had been tagged *questionable*. Because these utterances immediately followed student action

that presumably warranted correction, this subset of tutorial utterances served as the basis for comparing corrective tutorial strategies.

Overall, the forty-three tutoring sessions were effective, yielding a mean 5.9% learning gain from pretest to posttest across all participants. This difference is statistically significant (p=0.038, t-test with pooled variance, 42 *DF, SD=0.18*), though displaying a modest effect size of 0.33. For this study, cognitive benefit as well as motivational benefit were considered. Students rated their own self-confidence regarding the subject matter significantly higher, 12.1% on average, after the tutoring session than before (p=0.0021, t-test with pooled variance, 42 *DF, SD=0.24)* with effect size 0.5.

As in Study I, the student outcomes of learning gain and self-confidence gain for each participant were partitioned into binary categories of *High* and *Low* based on the median gain scores of all participants. Multiple logistic regression was then applied to determine whether a relationship existed between corrective tutorial strategy and student outcomes.

The results suggest that different types of tutorial feedback are related to different cognitive and motivational student outcomes. The analyses revealed the following results:

- Purely cognitive feedback was more often associated with high student learning gain than cognitive feedback that had an additional explicit component of *praise*. After accounting for the effects of pretest score and incoming confidence rating by including these as predictors in a logistic regression model with learning gain as the response variable,[2] observations in which the tutor used cognitive feedback plus praise were associated with 40% lower likelihood of high learning gain than

[2] These variables are included as predictors in all logistic regression models reported in this section in order to control for the influence of incoming knowledge level and confidence on the outcomes.

observations in which the tutor used purely cognitive feedback (p=0.001, significant logistic regression coefficient).

- Purely motivational dialogue moves, such as praise or reassurance, were associated with a greater gain in self-confidence among initially less-confident students. Observations in which the tutor employed a standalone motivational act were 300% as likely to be in the high confidence gain group as observations in which the tutor employed a purely cognitive statement or a cognitive statement combined with encouragement (p=0.039, significant logistic regression coefficient). On the other hand, these purely motivational acts on the part of the tutor were associated with 90% lower odds of high confidence gain in initially highly-confident students (p=0.04, significant logistic regression coefficient).
- The choice of tutor *positive cognitive feedback* was associated with 190% increased odds of the student experiencing high self-confidence gain compared to when tutors chose any other type of cognitive feedback (p=0.0057, significant logistic regression coefficient).

These results suggest that although some tradeoffs may exist between maximizing learning gains and strategies sometimes used to motivate the student, it may be possible to choose instructional strategies that enhance student motivation without sacrificing cognitive outcomes.

5.3 Study III: Student Control

A healthy level of student autonomy is thought to support increased motivation (*e.g.*, Dickinson, 1995). For this reason, student control during a tutoring session may be an important motivational component. One way to adjust student control in a one-on-one tutoring scenario is

for the tutor to take varying degrees of *initiative*. Results presented in this section explore whether there was difference in learning gains (measured by posttest score minus pretest score) or confidence gains between groups of students paired with tutors who naturally took significantly different levels of initiative.

Study III (Boyer, Phillips, Wallis, Vouk, & Lester, 2009) utilized the two most effective tutors from the prior two tutoring studies. There were sixty-one tutoring sessions distributed approximately equally between the tutors. From these sessions, fifteen were randomly selected for each tutor yielding a total of thirty sessions to be annotated for initiative using the annotation scheme described earlier.

Each Student-Initiative and Tutor-Initiative tag was associated with a duration of time over which that instance of the tutoring mode occurred. The sum (in minutes) of all Tutor-Initiative periods in a given tutorial session divided by the total time elapsed during the session yielded the percentage of the tutoring session that was spent in Tutor-Initiative mode. One tutor, referred to as the *moderate* tutor, took the initiative 55% of the time on average. The *proactive* tutor took the initiative an average of 73% of the time, constituting a significant difference in approach (p=0.029, t-test with pooled variances, 28 *DF*, *SD*=0.21). While one possible explanation for this difference could be that, despite the randomized assignment of students to tutors, the *moderate* tutor may have been assigned a group of students with a different level of preparedness than the *proactive* tutor, analysis of pretest scores suggests this confounding factor was not present. Average student pretest scores were 79.5% for the moderate tutor and 78.9% for the proactive tutor, yielding no evidence of a difference in student preparedness between the two treatment groups for the subset of students considered in the initiative annotation (p=0.764, t-test with pooled variances, 28 *DF*, *SD*=0.19).

For each participant, the cognitive outcome of *learning gain* was calculated as posttest score minus pretest score. The mean learning gain across each set of fifteen annotated student sessions was 6.9% for the moderate tutor and 6.0% for the proactive tutor, yielding no evidence of improved learning gains associated with a particular level of student control (p=0.895, t-test with pooled variances, 27 *DF*, *SD*=0.09).

It is reasonable to assume that the thirty sessions were representative of the larger data set in terms of tutor initiative because the subset was selected at random. Therefore, it is meaningful to consider all learning gains and assume each tutor took a sufficiently uniform approach across all tutoring sessions. The mean learning gain for *all* students tutored with the moderate approach was 6.9%, while the mean learning gain for the proactive tutor was 8.6%. In this larger set of learning gains, there is still no evidence that one tutor was more or less effective than the other (p=0.569, t-test with pooled variances, 58 *DF*, *SD*=0.11).

Student self-confidence gain was measured as the difference between post-survey and pre-survey score on an item that asked students to rate their confidence, on a scale of 0-100, in having the ability to learn the necessary course material for their introductory computer science class. A significantly different average confidence gain was found between student groups paired with the two tutors. Students who worked with the *proactive* tutor had an average confidence gain of less than one point from pre-survey to post-survey. On the other hand, students paired with the *moderate* tutor had an average confidence gain of more than six points, which is significantly higher (p=0.047, t-test with pooled variance, 28 *DF*, *SD*=6.5). This finding suggests the following hypothesis: within the two levels of tutor initiative considered here, affording the student more control may yield motivational benefit without sacrificing cognitive outcomes.

6. Discussion

Extensive research, including many active projects by computing education researchers and practitioners, are providing an emerging picture of the cognitive issues that arise on a student's path toward understanding computing. These results give rise to interventions aimed at addressing misconceptions, repairing mental models, and facilitating algorithmic thinking. In the same way, an understanding of student motivation may guide instructional interventions that enhance student engagement and encourage persistence in learning computing. The instructional strategies presented here were observed in naturalistic human-human tutoring sessions in introductory computer science. The task-oriented tutorial sessions involved experienced human tutors and novice computer science students who were working to solve a programming exercise. The tutors and students communicated remotely through textual dialogue. Textual communication is representative of much of the instructional discourse that takes place in today's computer science learning environments; therefore, results concerning the instructional strategies used in remote textual human-human tutoring are meaningful for other areas such as online message boards, correspondence with teaching assistants, and the design of educational systems.

6.1 Discussion of Related Work

The results reported here suggest that structural differences in tutorial dialogue can be observed depending on whether students display high or low initial confidence related to the computing task at hand. In addition, findings suggest the hypothesis that certain instructional strategies, such as positive feedback, are associated with increased student confidence gains. This result reinforces prior research suggesting that during an introductory programming course, in order to

increase self-efficacy, students benefit from frequent small successes and feedback (Ramalingam, LaBelle, & Weidenbeck, 2004). In the tutorial case, the reason for providing positive feedback even following potentially incorrect student problem-solving action may be that the positive feedback can be followed by correction, and this *indirect correction* is an example of a potentially beneficial politeness strategy in which the tutor avoids direct confrontation regarding a student's mistake (*e.g.*, Porayska-Pomsta & Pain, 2004; Wang, Johnson, Rizzo, Shaw, & Mayer, 2005).

The final study involved analysis of tutorial dialogue structure to investigate the impact of instructor initiative. The study found that students who were allowed more control experienced higher gains in self-confidence, with no significant difference in learning gain. This is an important preliminary finding, since an increased sense of control is hypothesized to be beneficial for student motivation (Dickinson, 1995). While some motivational and cognitive goals are known to be at odds with one another (*e.g.*, Lepper, Woolverton, Mumme, & Gurtner, 1993), future work on student self-confidence could clarify how levels of student control can enhance motivation without sacrificing cognitive outcomes.

6.2 Threats to Validity

The three studies presented here are exploratory in nature and, as such, are not intended to confirm specific hypotheses regarding the impact of instructional strategies on student motivation. The primary limitation of the studies stems from two sources: 1) the absence of control groups, and 2) the abundance of factors that were allowed to vary across treatments. It is plausible that factors other than those explored here are responsible for the association between tutorial strategies and student motivation. However, the work presented here has revealed hypotheses that can inform future controlled experiments.

Another threat to validity involves the potential error when applying the heuristic for problem-solving action correctness in Study II. Based on the automatic application of a small set of rules, the tags of *Questionable* or *Promising* with regard to student actions may not correspond well to the judgment the tutors made in real time. This confound is a potential source of error whose magnitude cannot be assessed with the data available. Future studies will feature manual tagging of the student problem-solving actions to avoid the challenges presented by the rule-based heuristic tags.

Finally, handcrafted tests and surveys comprise the sources of quantitative data in this mixed methods research. For Studies I and II, the tests did not undergo any formal validation, and for Study III the evaluation was limited to review by a panel of three subject matter experts. The surveys, while based on widely-used instruments to measure student self-confidence, have not themselves undergone validation studies.

7. Conclusion

One-on-one human tutoring is a viable research tool for studying fine-grained instructional approaches and their effects on individual students. Our findings suggest that in computer science education, instructional strategy may have an appreciable effect not only on purely cognitive outcomes such as learning gain, but also on important motivational outcomes such as self-confidence. These results, which are based on three separate exploratory studies of naturalistic human-human tutoring in introductory computer science, provide early clues as to the impact of different instructional strategies on computer science students.

Future work will involve integrating the corpora from the three separate studies. It is likely that, to the extent these data sets can be considered homogeneous, combining them can strengthen and clarify the results presented here. Another area for future work includes

conducting focused experiments to test the hypotheses that emerged from these exploratory studies. In future experiments, enhancements to the current methodology, such as manually tagging student problem-solving actions, could strengthen the findings. In addition, other work on the motivational components of *curiosity* and *challenge* (Lepper, Woolverton, Mumme, & Gurtner, 1993) suggest that these aspects of student motivation should also be investigated along with confidence and control in the context of computer science education.

8. Acknowledgements

The authors wish to thank the following for insightful discussions and support in preparing this manuscript: Scott McQuiggan and the members of the Intellimedia Center for Intelligent Systems at NC State University, Carolyn Miller, and Tiffany Barnes. Thanks to the software development team of August Dwight and Taylor Fondren whose outstanding undergraduate research projects resulted in the software that facilitated these tutoring studies, and to the Realsearch Group at NC State University for their project development support.

This work has been supported in part by the National Science Foundation through grants REC-0632450, IIS-0812291, the STARS Alliance grant CNS-0540523, and an NSF Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Support has also been provided by North Carolina State University through the Department of Computer Science and the Office of the Dean of the College of Engineering.

9. References

Bandura, A. (2006). Guide for constructing self-efficacy scales. In F. Pajares, & T. Urdan (Eds.), *Self-efficacy beliefs of adolescents* (pp. 307-337). Greenwich, Connecticut: Information Age Publishing.

Barker, L. J., & Garvin-Doxas, K. (2004). Making visible the behaviors that influence learning environment: A qualitative exploration of computer science classrooms. *Computer Science Education, 14*(2), 119-145.

Ben-Ari, M. (1998). Constructivism in computer science education. *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education,* 257-261.

Bloom, B. S. (1956). *Taxonomy of educational objectives: The classification of educational goals*. New York: David McKay.

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.

Boyer, K. E., Dwight, A. A., Fondren, R. T., Vouk, M. A., & Lester, J. C. (2008). A development environment for distributed synchronous collaborative programming. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education,* Madrid, Spain. 158-162.

Boyer, K. E., Phillips, R., Wallis, M., Vouk, M., & Lester, J. (2008). Balancing cognitive and motivational scaffolding in tutorial dialogue. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems,* Montreal, Canada. 239-249.

Boyer, K. E., Phillips, R., Wallis, M., Vouk, M., & Lester, J. (2009). The impact of instructor initiative on student learning through assisted problem solving. *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education,* Chattanooga, Tennessee. 14-18.

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),* Marina del Rey, California. 365-372.

Cade, W., Copeland, J., Person, N., & D'Mello, S. (2008). Dialog modes in expert tutoring. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems,* Montreal, Canada. 470-479.

Cameron, J., & Pierce, W. D. (1994). Reinforcement, reward, and intrinsic motivation: A meta-analysis. *Review of Educational Research, 64*(3), 363.

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.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*(1), 37-46.

Creswell, J. W., & Clark, V. L. P. (2006). *Designing and conducting mixed methods research*. Thousand Oaks, California: Sage Publications.

Davis, M. H. (1983). Measuring individual differences in empathy: Evidence for a multidimensional approach. *Journal of Personality and Social Psychology, 44*(1), 113-126.

Deci, E. L., Koestner, R., & Ryan, R. M. (2001). Extrinsic rewards and intrinsic motivation in education: Reconsidered once again. *Review of Educational Research, 71*(1), 1-27.

Dickinson, L. (1995). Autonomy and motivation: A literature review. *System, 23*(2), 165-174.

Elliot, A. J., & McGregor, H. A. (2001). A 2 x 2 achievement goal framework. *Journal of Personality and Social Psychology, 80*(3), 501-519.

Forbes-Riley, K., Litman, D., Huettner, A., & Ward, A. (2005). Dialogue-learning correlations in spoken dialogue tutoring. *Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIEd),* Amsterdam. 225-232.

Graesser, A. C., Person, N. K., & Magliano, J. P. (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology, 9*(6), 495–522.

Guzdial, M., & Tew, A. E. (2006). Imagineering inauthentic legitimate peripheral participation: An instructional design approach for motivating computing education. *Proceedings of the Second International Computing Education Research Workshop (ICER),* Canterbury, United Kingdom. 51-58.

Jurafsky, D., & Martin, J. H. (2008). *Speech and language processing*. Upper Saddle River, New Jersey: Pearson.

Keller, J. M. (1983). Motivational design of instruction. In C. M. Reigeluth (Ed.), *Instructional-design theories and models* (pp. 386-434) Hillsdale: Erlbaum.

Lane, H. C., & VanLehn, K. (2005). Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education, 15*(3), 183-201.

Layman, L., Williams, L., & Slaten, K. (2007). Note to self: Make assignments meaningful. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education,* Covington, Kentucky. 459-463.

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. In S. P. Lajoie, & S. J. Derry (Eds.), *Computers as cognitive tools* (pp. 75-105). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Lister, R., Berglund, A., Box, I., Cope, C., Pears, A., Avram, C., et al. (2007). Differing ways that computing academics understand teaching. *Proceedings of the Ninth Australasian Conference on Computing Education,* 97-106.

Machanick, P. (2007). A social construction approach to computer science education. *Computer Science Education, 17*(1), 1-20.

Porayska-Pomsta, K., & Pain, H. (2004). Providing cognitive and affective scaffolding through teaching strategies: Applying linguistic politeness to the educational context. *Proceedings of the 7th International Conference on Intelligent Tutoring Systems,* Alagoas, Brazil. 77-86.

Ragonis, N., & Hazzan, O. (2008). Tutoring model for promoting teaching skills of computer science prospective teachers. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education,* Madrid, Spain. 276-280.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and mental models in learning to program. *Proceedingss of the International Conference on Innovation and Technology in Computer Science Education (ITiCSE),* Leeds, United Kingdom.

Rebolledo-Mendez, G., Boulay, B., & Luckin, R. (2006). Motivating the learner: An empirical evaluation. *Proceedings of the 8th International Conference on Intelligent Tutoring Systems,* Jhongli, Taiwan. 545-554.

Slaten, K. M., Droujkova, M., Berenson, S., Williams, L., & Layman, L. (2005). Undergraduate student perceptions of pair programming and agile software methodologies: Verifying a model of social interaction. *Proceedings of AGILE,* Denver, Colorado. 323-330.

Soh, L. K., Samal, A., & Nugent, G. (2007). An integrated framework for improved computer science education: Strategies, implementations, and results. *Computer Science Education, 17*(1), 59-83.

Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., et al. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics, 26*(3), 339-373.

Tan, J., & Biswas, G. (2006). The role of feedback in preparation for future learning: A case study in learning by teaching environments. *Proceedings of the 8th International Conference on Intelligent Tutoring Systems,* Jhongli, Taiwan. 370-381.

The Eclipse Foundation. (2009). *Eclipse home page.*, 2009, from www.eclipse.org

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, N., Johnson, W. L., Rizzo, P., Shaw, E., & Mayer, R. E. (2005). Experimental evaluation of polite interaction tactics for pedagogical agents. *Proceedings of the 10th International Conference on Intelligent User Interfaces,* San Diego, California. 12-19.

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. *Proceedings of the First International Computing Education Research Workshop (ICER),* Seattle, Washington. 13-24.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education, 12*(3), 197-212.

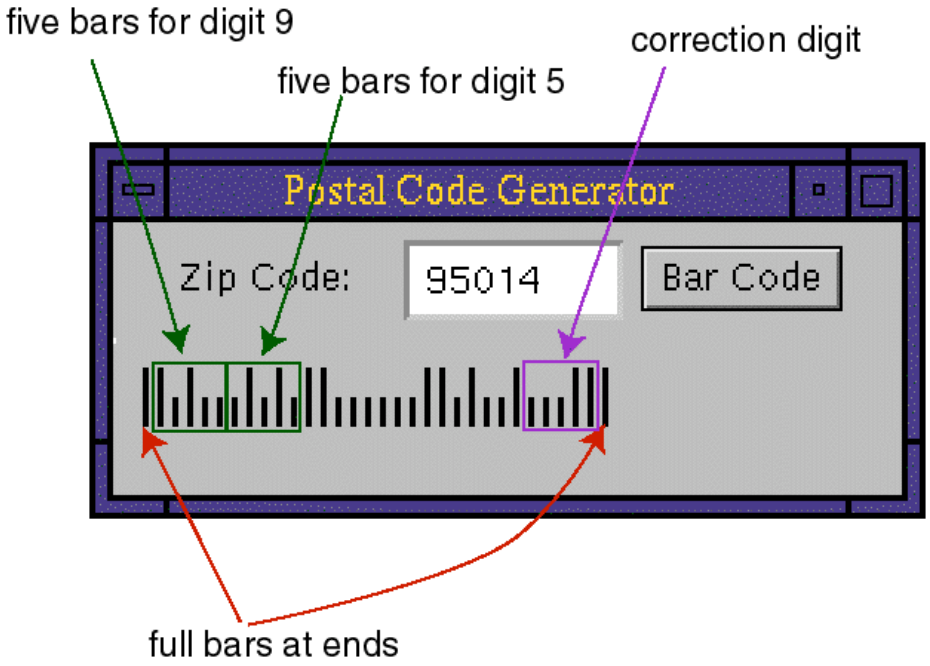
Wolfe, J. (2004). Why the rhetoric of CS programming assignments matters. *Computer Science Education, 14*(2), 147-163.

Appendix A

Programming Exercise for Studies I and II

The Problem:
For faster sorting of letters, the United States Postal Service encourages companies that send large volumes of mail to use a bar code denoting the ZIP code. Using the skeleton GUI program provided for you, you will complete this lab with code to actually generate the bar code for a given zip code.

More About Bar Codes:
In postal bar codes, there is a full-height frame bar on each end (and these are drawn automatically by the program provided for you; you don't have to write code to draw these). Each of the five encoded digits is represented by five bars. The five encoded digits are followed by a correction digit.



The Correction Digit
The correction digit is computed as follows: Add up all digits, and choose the correct digit to make the sum a multiple of 10. For example, the ZIP code 95014 has sum of digits 19, so the correction digit is 1 to make the sum equal to 20.

What's Already Written?
You can see what parts of this program are already written by running the file Main.java. When you do, you should see output like the image below, with a blank zip code slot. You can enter a zip code, and you should see that no bar code is generated (except the first and last full bars which are required for all bar codes).

What's Your Task?

Your job is to take this five-digit zip code and use it to generate a bar code. The PostalFrame class is the one which handles this task. The three methods which you must complete are:

extractDigits()
calculateAndDrawCDigit()
drawZIPCode()

For extractDigits(), you will need to add a private variable to the class which stores the zip code as separate digits.

Some Helpful Information

- If you can't remember how to do something with the software, please refer to the reference sheet on your desk.

- This lab involves a package named postal. This package contains classes Bar, FullBar, PostalBarCode, and SmallBar. The reason these classes are grouped into a package, is that the classes of the postal package logically belong together to accomplish a task. Whenever you need to use things from one package outside of that package, you just import the package. This has already been done for you in Main and PostalFrame – you will see the statement import postal.* at the top. In addition to code already provided, you will need to call methods in the PostalBarCode class from your PostalFrame class to draw full and small bars.

- Each digit of the ZIP code and the correction digit are encoded according to the following table (each digit has five bars -- a zero is a half bar and a one is a full bar). This scheme represents all combinations of two full and three half bars.

Digit	
0	1 1 0 0 0
1	0 0 0 1 1
2	0 0 1 0 1
3	0 0 1 1 0
4	0 1 0 0 1
5	0 1 0 1 0
6	0 1 1 0 0
7	1 0 0 0 1
8	1 0 0 1 0
9	1 0 1 0 0

Appendix B

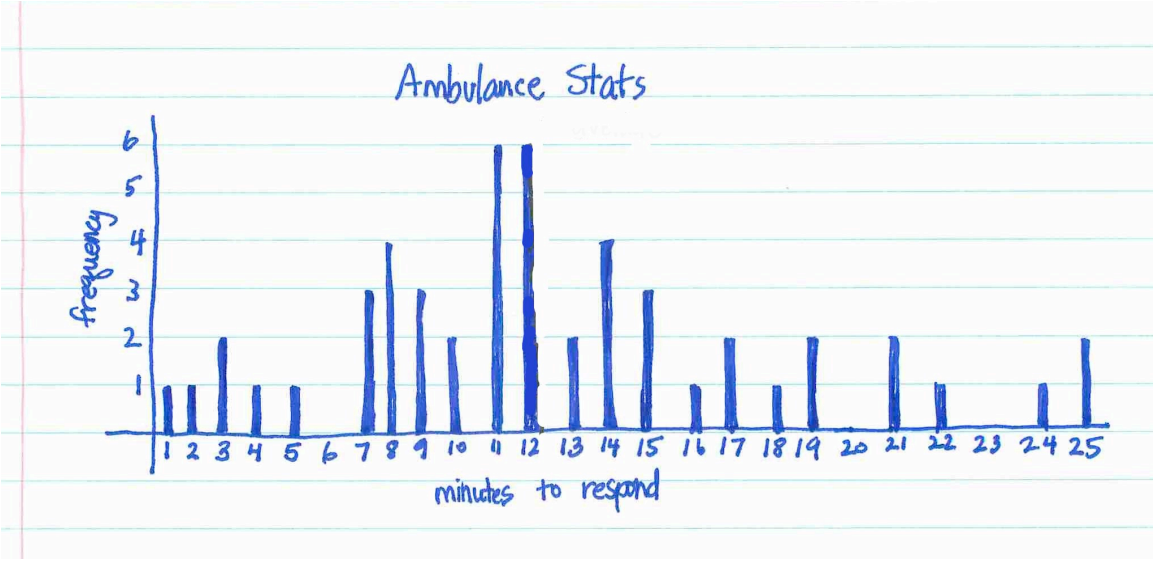
Programming Exercise for Study III

To help ensure the safety of their residents, the Waimea County Emergency Response office is re-assessing their ambulance dispatch system. A study has already been conducted to gather data about the ambulance response times to 911 calls. You have been hired to analyze this data and help the emergency response office answer some questions about how quickly their ambulances are able to reach people in need. You'll be taking over for Maddie, the previous developer who was recently promoted.

Maddie already completed the class called Ambulance.java, which is a driver for the whole program (it contains the main method). She also completed AmbulanceGUI.java, which is used for displaying the ambulance response times graphically. You just need to complete a few methods in the AmbulanceData class in order to finish this project!

1. In the AmbulanceData class, you must complete the method plotTimes() so that all the ambulance response times in the parameter array (arrayToPlot) are displayed on a graph. Maddie already created the method outline with some comments, so you'll just need to read her comments and fill in the method.

   Maddie had an intern draw a graph by hand for the response times. This way, you know what the output of your program is supposed to look like. The x-axis is how many minutes an ambulance took to respond, the y-axis is a count of how many of the response times in the data set took that long. For instance, there were three ambulance responses that took 7 minutes.



2. The department is considering replacing its aging fleet with new ambulances. Because of the county's tight budget, these would be slightly slower ambulances than the current fleet but the

county could afford more ambulances overall. The staff believe the effects of this change would be:

- On all response times below 5 minutes, the new fleet would take 1 minute longer to respond.
- On all response times above 18 minutes, the new fleet would take 4 fewer minutes to respond.
- Other response times would remain the same.

Complete the method newFleetProjections() which creates a new array of hypothetical response times given the above effects of the new fleet. You will need to create a new array because you must not overwrite the true response times in the original array.

3. There is more analysis work than Maddie originally thought, so one of your colleagues, Shannon, is writing a set of methods that perform the statistical analysis so your group can give a detailed report to the Waimea County authorities. Shannon's code needs to be able to pass an array of unsorted times to a sortArray method, and get back an array of sorted times. Write a method called sortArray in the AmbulanceData class. The sortArray method should take an array of doubles as a parameter, and return a sorted ascending version of the parameter array *without overwriting the contents* of the original array.

   The next page has some details of how your sort method should work.

Appendix C

Confidence and Attitude Survey from Study III

Please rate how certain you are that you can do each of the things described below by writing the appropriate number.

Rate your degree of confidence by recording a number from 0 to 100 using the scale given below:

0	10	20	30	40	50	60	70	80	90	100
Cannot do at all					Moderately can do					Highly certain Can do

	Confidence (0-100)
Learn Computer Science.	_____
Learn CSC 116 course material.	_____
Complete a *simple* programming exercise on my own.	_____
Complete a *challenging* programming exercise on my own.	_____
Complete a *challenging* programming exercise if I am in a lab where a TA is available to help me.	_____
Explain for-loops to others well.	_____
Explain arrays to others well.	_____
Explain method calls to others well.	_____
Use for-loops in a programming exercise correctly and effectively.	_____
Use arrays in a programming exercise correctly and effectively.	_____
Make method calls in a programming exercise correctly and effectively.	_____

Please rate the degree to which you agree or disagree with the following statements:

	Not At All		Moderate		Very Much
I usually enjoy CSC 116 course material.	○	○	○	○	○
I usually find CSC 116 exercises challenging.	○	○	○	○	○
I understand for-loops.	○	○	○	○	○
I understand arrays.	○	○	○	○	○
I am experienced using the eclipse development environment.	○	○	○	○	○
I am experienced with communicating through typed instant messages.	○	○	○	○	○

Appendix D

Pre/Post Test Excerpt from Study III

1. Write a chunk of Java code to accomplish each of these tasks:

   a. Declare an array of integer type and give it an initial size of 100.

   b. Test the i^{th} element of the array you declared in part a of this question and print "true" if the element is equal to 5 and "false" otherwise. Assume that i has already been declared and initialized.

   c. Set the i^{th} element of the array you declared in part a of this question to be 5. Again, assume that i has already been declared and initialized.

2. A separate class named `MyClass` defines a `printValue` method as follows:

```
public static void printValue(float x) {
      // function body here
}
```

   In the following `PrintAllValues` method you want to call the method `MyClass.printValue` on every element of the array `myArray`. Fill in the blanks below to do this.

```
public void PrintAllValues() {
      float [] myArray = new float[20];

      for (_____________;_____________;_____________) {
            MyClass.printValue(_____________);
      }
}
```

3. Write a piece of Java code that prints "Cowabunga!" exactly 73 times. System.out.println can be used to print the string.

4. In a Java program, an array named `firstArray` of type `int` has been created and initialized. Write a line of Java code to create an array named `secondArray` that is the same size and same type as `firstArray`. The contents of `secondArray` do not need to be initialized to be the same as the contents of `firstArray`.

5. Complete the following Java method so that it returns the average of all the elements in the array `myArray`.

```
public double returnAverage(double [] myArray) {
   double average = 0;

   return average;
}
```