Int J Artif Intell Educ (2017) 27:101–129
DOI 10.1007/s40593-015-0071-y

ARTICLE

Shifting the Load: a Peer Dialogue Agent that Encourages its Human Collaborator to Contribute More to Problem Solving

Cynthia Howard[1] · Pamela Jordan[2] · Barbara Di Eugenio[3] · Sandra Katz[2]

Published online: 27 October 2015

Abstract Despite a growing need for educational tools that support students at the earliest phases of undergraduate Computer Science (CS) curricula, relatively few such tools exist–the majority being Intelligent Tutoring Systems. Since peer interactions more readily give rise to challenges and negotiations, another way in which students can become more interactive during problem solving, we created an artificial peer collaborator to determine its value for aiding CS students. Central to its development was the notion that it should monitor the student's collaborative behavior and attempt to guide him/her towards more productive behavior. In prior work, we found that *initiative shifts* correlate with both Knowledge Co-Construction (KCC) and learning and are potentially easier to model as an indicator of productive collaboration in instructional software. In this paper, we describe a unique peer dialogue agent that we created to test the effects of tracking and reacting to initiative shifts.

✉ Pamela Jordan
pjordan@pitt.edu

Cynthia Howard
howardcy@lewisu.edu

Barbara Di Eugenio
bdieugen@uic.edu

Sandra Katz
katz@pitt.edu

[1] Computer and Mathematical Sciences Department, Lewis University, One University Parkway, Romeoville, IL 60446-2200, USA

[2] Learning Research and Development Center, University of Pittsburgh, 3939 O'Hara St, Pittsburgh, PA 15260, USA

[3] Department of Computer Science, University of Illinois at Chicago, 1120 Science & Engineering Offices (MC 152) 851 South Morgan, Chicago, IL 60607, USA

While our study did not find differences in learning gains when comparing agents that do and do not track and react to initiative shifts, we did find that students do learn when interacting with the agent and that attempting to influence initiative taking did make a difference. This suggests that by tracking initiative shifts, the agent was able to detect times when the student had been letting the agent do most of the "deep thinking" and that the agent's tactics for encouraging the student to begin taking the initiative again were helpful.

Keywords Peer agent · Collaborative problem solving · Collaborative dialogue · Computer science education

Introduction

Introductory data structures and their related algorithms is one of the core components of Computer Science education. A deep understanding of this topic is essential to a strong Computer Science foundation, as attested by the curricula promoted by national and international professional societies (AA.VV. 2001, 2007, 2013; Scime 2008). Computer Science is of enormous strategic interest, and it is projected to foster vast job growth in the next few years (AA. VV. 2014; Brienza 2012); however, in the United States there is a dearth of qualified professionals to step into these openings. Whereas in the last few years the number of CS majors has robustly increased (Zweben and Bizot 2015), reversing the dramatic downward trend from the middle '2000s, CS programs nonetheless often experience exceedingly high rates of attrition (Beaubouef and Mason 2005; Katz 2006; DeClue et al. 2011; Falkner and Falkner 2012; Porter et al. 2013; Beyer 2014). Thus, educational tools that help students at the earliest phases of undergraduate Computer Science curricula, where attrition most often takes place, could be beneficial.

However, relatively few advanced educational software systems address Computer Science topics; for example, teaching a specific programming language like LISP (Corbett and Anderson 1990) or database concepts (Mitrović et al. 2004; Latham et al. 2014).[1] Even fewer systems address CS foundations, including: AutoTutor (Graesser et al. 2004), which addresses basic literacy but not data structures or algorithms; ProPL (Lane and VanLehn 2003) which helps novices design their programs, by stressing problem-solving and design skills; ILMDA (Soh 2006), which has a module on recursion; and iList/ChiQat (Fossati et al. 2009, 2015; Alzoubi et al. 2014), which helps students learn linked lists, and more recently, recursion. Boyer et al. (2011) and Ezen-Can and Boyer (2013) carefully model tutorial dialogues from an introductory course on Java; recently, those models have been embedded in an Intelligent Tutoring System on Java programming (Ezen-Can and Boyer 2015).

The vast majority of these software systems are Intelligent Tutoring Systems, and when the student requests help the system acts as an authority on the subject matter

[1] Additional research focuses on ITSs that address basic computer skills such as word processing or using spreadsheets (Wang et al. 2015). Those topics are not considered foundational for a CS education.

and the problem-solving approach, just as a human tutor would. In a human-human tutoring situation, students appear to be less willing to argue with or challenge the tutor's knowledge or approach. The ICAP (Interactive-Constructive-Active-Passive) framework suggests that the more active the student is in a learning activity, the more beneficial the activity is to the student's learning outcome (Chi 2009). But the social convention of respecting the tutor's authority, combined with the fact that ITSs are not equipped to argue or negotiate with students who might be willing to flout convention (and perhaps nor should they be so equipped), means that one way in which students can become more interactive in problem solving is unavailable. Thus as part of our research agenda, we set out to explore whether a peer collaborative problem-solving agent could improve students' performance and understanding of CS data structures. Our longer-term goal is to compare whether peer interactions offer any benefits beyond what current ITSs can, also taking into account the limitations of current technology (Harsley 2015).

Peer-to-peer interactions are notably different from those of expert(tutor)-novice(student) pairings, especially with respect to the richness of the problem-solving deliberations and negotiations. A hint at the difference between the two types of interactions can be gleaned from contrasting the excerpt from Fig. 1 to the one in Fig. 2. In Fig. 1, the two peers often contribute content which, from a problem-solving point of view, is new with respect to what the partner suggests and is not requested by the partner–see P4 for peer M, and P3, P5, P7 and P8 for peer C. In contrast, Fig. 2 includes a typical excerpt from one of 54 one-on-one human tutoring sessions we collected, on the same introductory Computer Science topics (Fossati et al. 2009; Di Eugenio et al. 2009). Student 102 only has two short turns, the first in T4 when he answers the tutor's question at T3. Only once, in T6, does student 102 contribute content that is new and is not in response to a tutor question.

In addition to the differences between tutoring and successful collaborative problem-solving dialogues, which illustrate that students contribute more towards problem solving with the latter than the former, true collaborative problem solving means that when one student is stuck or weak in a particular area, the other can possibly take over and move the problem solving forward and vice versa; if both are

```
P1]   47:00 M: ok so that's a picture of what we want...
               now we need to decide if code actually does that..
P2]   47:11 C: yeah
P3]   47:41 C: first it created a list of 5 nodes,
               with values 5,4,3,2, and 1
P4]   48:08 M: yes, then it removes the 5 and puts in the 88 in
               it's place
P5]   48:25 C: i think it removes 1 and puts 88 in its place
P6]   48:29 C: <in code selects line 20>
P7]   49:10 C: yeah, teh way that "link()" function works is
               basically like an "insertBefore()"
P8]   49:45 C: so it builds the list backwards
```

Fig. 1 Peer-to-Peer interaction excerpt

```
T1]   26:05 TUT: we found the "a", and we found the node before "a"
                 which is what we really need.
T2]   30:13 TUT: so the surgery is pretty simple.
T3]   34:06 TUT: we are going to just do this which is ...
                 what is that?
T4]   39:15 102: s.next
T5]   40:24 TUT: yeah it's the next of "s".
T6]   43:15 102: equals t.next
T7]   44:28 TUT: is the next of t
T8]   48:01 TUT: and then
T9]   49:05 TUT: ooh we're done.
```

Fig. 2 A tutoring interaction excerpt

stuck they can work together to try to overcome the impasse. Collaborative software systems that focus on learning CS concepts include Constantino-González et al. (2003) and Soller (2004). However, they aim at mediating the interactions between two human peers, not at playing the role of a peer software agent interacting itself with a human.

In this paper, we present, KSC-PaL, a collaborative problem-solving dialogue agent for CS data structures that is unique to CS education, and education in general, because it engages in a one-on-one problem-solving peer interaction with a student and can interact verbally, graphically and in a process-oriented way.[2] The peer status of the agent does seem to encourage students to look more critically at its contributions, yet it does not leave them to flounder indefinitely when they get stuck. But because it is designed to engage in collaborative problem solving instead of tutoring, its behavior is different from that of a peer tutor or a peer tutee/teachable agent (e.g., Cai et al. 2014).

In this paper, we describe the CS education concerns that influenced the design of the agent, how those concerns were translated into computational models on the basis of a human-human data collection, and an experiment in which we compare two different agent interaction policies. The control version of the agent does not monitor the interaction for good collaborative behavior on the student's part, while the experimental version does monitor the interaction. When the monitored behavior falls below a certain threshold, the experimental version of the agent attempts to alter the student's collaborative behavior using a number of discourse moves. We analyze whether students learn when interacting with the agent; whether the collaborative behavior of interest–namely, taking task initiative–correlates with learning, as had been found in the human-human peer interactions we had analyzed; and whether the experimental agent is able to alter the student's behavior.

[2]This paper is a vastly expanded version of the conference paper Kersey (Howard) et al. (2010). The current paper provides the CS education motivations for the work, expands on the corpus analysis results that informed the agent design, and provides far more details on how the agent tracks and shifts task initiative. Additionally, the experiment results are analyzed at much greater depth.

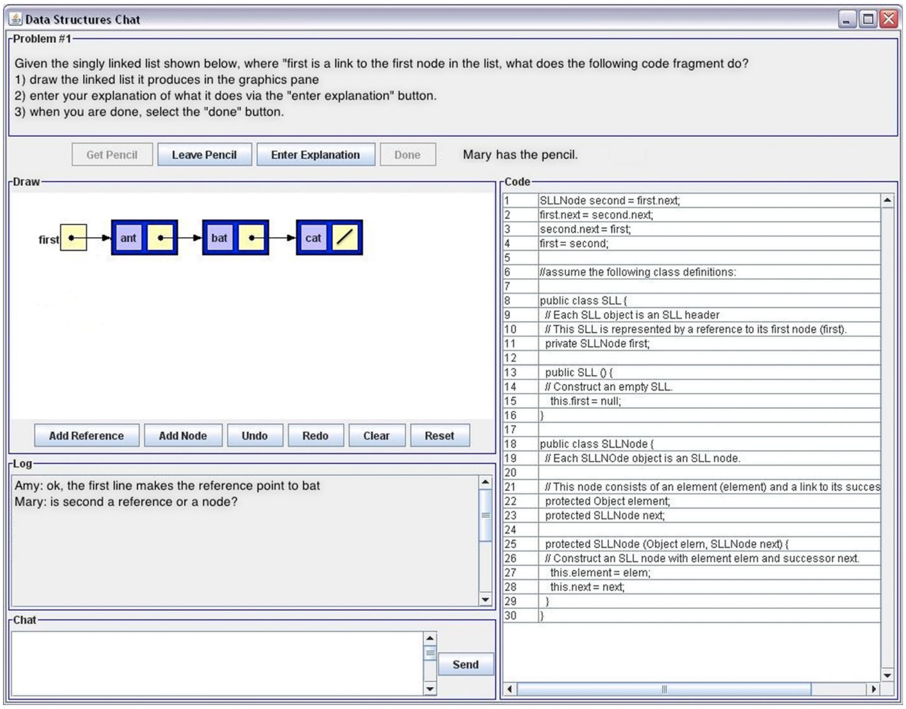
Learning Challenges in Introductory Computer Science

In earlier work, we collected five face-to-face interactions and fifteen computer-mediated interactions between two peer students who were collaboratively working to solve problems about three different data structures (Kersey (Howard) et al. 2008, 2009; Kersey (Howard) 2009; Howard et al. 2015). The problems were presented in the same order in which they are typically introduced in a CS course: linked lists, stacks and binary search trees (BSTs). These data structures are typically presented to students in one or two courses taken either during spring of freshman year, or fall of sophomore year.

To our knowledge, systematic studies of why difficulties arise for specific data structures do not exist. Most research relating cognitive science and CS either attempts to holistically inform the whole CS curriculum in the classroom (Ben-Ari 1998; Lister and Leaney 2003; Scott 2003; Fuller et al. 2007), or examines how novices approach programming (Soloway and Spohrer 1988; Winslow 1996; Mead et al. 2006; Renumol et al. 2010).

In general, CS educators consider stacks the easiest of the three data structures under consideration, followed by lists and then BSTs. There are several possible causes for the relative difficulty of these data structures. First, lists and BSTs are the first *recursive* data structures CS majors encounter, namely, structures whose sub-structures still satisfy the same definition. The cognitive difficulty of recursion in CS was studied early on, within the context of model-based tutoring systems (Pirolli and Anderson 1985; Pirolli and Recker 1994). People appear to have inherent difficulties with recursion, well beyond CS. In fact, whereas self-embedding, recursive definitions and processes are pervasive in mathematics and in CS (Wing 2006), they also occur in natural language, as in: *[The fact [that the shepherd said [that the farmer has given the book to the child] to the police] was to be expected]*. Evidence has accumulated for almost 50 years that people have difficulties with recursion in language (Miller and Chomsky 1963; Anderson 1976; Karlsson 2007). BSTs add yet another level of complexity on top of recursion, since they are two-dimensional, as opposed to stacks and lists, which are one-dimensional. However, our analyses of the human-human peer interactions in which students worked on solving these problems suggests that collaboration on the problem types we explored may have been beneficial only for linked lists (Kersey (Howard) 2009).

A second reason for the difficulties encountered by students with these introductory data structures is that, as in physics or chemistry (Kozma 2003; Meltzer 2005), students must be able to develop and use multiple mental representations of the same concept. For example, in textbooks, data structures are presented verbally, graphically and in a process-oriented way, via code or pseudo-code (Hundhausen et al. 2002). In the face-to-face interactions we collected and examined, students not only talked about the problems but, without prompting, drew pictures of the data structures involved and marked up the associated code. Thus, the interface for the computer-mediated environment for the data collection and for KSC-PaL was designed to support the same. Figure 3 shows two representations of a list: graphical in the drawing window on the left, and code in the window on the right.



Given the singly linked list shown below, where "first is a link to the first node in the list, what does the following code fragment do?
1) draw the linked list it produces in the graphics pane
2) enter your explanation of what it does via the "enter explanation" button.
3) when you are done, select the "done" button.

Mary has the pencil.

Amy: ok, the first line makes the reference point to bat
Mary: is second a reference or a node?

```
SLLNode second = first.next;
first.next = second.next;
second.next = first;
first = second;

//assume the following class definitions:

public class SLL {
 // Each SLL object is an SLL header
 // This SLL is represented by a reference to its first node (first).
 private SLLNode first;

 public SLL () {
 // Construct an empty SLL.
  this.first = null;
}

public class SLLNode {
 // Each SLLNOde object is an SLL node.

 // This node consists of an element (element) and a link to its succes
 protected Object element;
 protected SLLNode next;

 protected SLLNode (Object elem, SLLNode next) {
 // Construct an SLL node with element elem and successor next.
  this.element = elem;
  this.next = next;
 }
}
```

Fig. 3 The student's interface in KSC-PaL

Each dyad in the computer-mediated data collection effort was presented with three problems on linked lists, one on stacks, and one on BSTs. The final problem set was either code explanation problems or *debugging*; namely, error diagnosis problems. For explanation problems, the subjects are asked to explain what a piece of code does and draw the resulting data structure. To explain what the code does, students need to *simulate* the code; that is, trace what the code does. In the debugging problems, the subjects are told the code has one or more mistakes and are asked to find the mistake(s). They again need to simulate the code to find the mistake (the subjects know the code is syntactically correct; namely, it conforms to the grammar for the specific programming language). In short, explanation problems differ from debugging problems because the subjects know that in explanation problems the code is semantically correct, but that in debugging problems the code is semantically incorrect. At least some of the same mental processes apply to both problems, but debugging problems are likely to require more complex cognitive processes (Yoon and Garcia 1998; Xu and Rajlich 2004; McCauley et al. 2008) – as (McCauley et al. 2008) note, "Debugging is an important skill that continues to be both difficult for novice programmers to learn and challenging for computer science educators to teach." (p. 67)

To develop the five problems we used in the earlier study of human-human interactions and the later study of agent-human interaction, which used two of these problems and is the focus of this paper, we reviewed program examples and problems in textbooks. We altered the problems in two other ways in order to encourage collaboration: each problem statement requested that students come to an agreement on the solution and when they agreed they had solved it, to enter their explanation. Initially we created only debugging problems which included both syntactic and semantic bugs. Upon examining the first three face-to-face interactions, we found that students were focusing too much on trying to find syntactic bugs and overlooking semantic bugs. So we revised the problems and removed all syntactic bugs. Additionally, we added explanation problems in order to get more insight into what students were thinking and to encourage them to simulate the code.

Attrition and Underrepresentation in Computer Science Education

In addition to our interests in addressing the difficulties any student entering CS encounters when learning data structures, we are also concerned about low female enrollment and retention in CS programs. Despite the recent surge in enrollment in CS, women and minorities are still underrepresented: for example, in 2013, women earned only 13 % of bachelor's degrees in CS, African-Americans 3.8 % and Hispanics 6 % –African-Americans and Hispanics made up 15 % and 14 % of undergraduate students in the U.S., respectively (Zweben and Bizot 2015; Monge et al. 2015). Among the many suggestions from researchers who explore possible reasons for low participation by females (Fisher et al. 1997; Gürer and Camp 2002; Wilson 2002, Katz et al. 2003, 2006; Barker and Garvin-Doxas 2004) is the behaviors of students and instructors in CS classrooms. Barker and Garvin-Doxas (2004) explored the discourse in multiple CS classrooms and found communication patterns that were indicative of "an impersonal environment with guarded behavior" and of participants creating and maintaining a hierarchy based on programming experience that resulted in competitive behaviors. Since females entering CS may have less programming experience (Gürer and Camp 2002), this seemingly hostile environment could be one possible cause for low retention (Wilson 2002). An example of one negative communication pattern was a strong tendency for students to correct the instructor's syntactic errors when a concept was the focus of instruction. The instructor then accepted the correction and made statements that explicitly strengthened the attitude that programming skill reflects ability rather than practice (Barker and Garvin-Doxas 2004). Regarding programming skill as a measure of innate ability seems to be at the core of the negative communication patterns and is reminiscent of the varying achievements found when intelligence is considered a fixed trait rather than a potential that can be developed (Hunt 1961; Blackwell et al. 2007).

We hypothesize that emphasizing semantics over syntax in data structures problems and encouraging collaboration could help counteract students' competitive behaviors and mistaken equating of programming skill with innate intelligence. In recent years, the paradigm of *pair programming* has been shown to increase retention of women and underrepresented minorities (Li et al. 2013; Reese et al. 2014). Pair programming is inherently collaborative, since in this paradigm two programmers work together on one workstation: one, the driver, writes code while the other, the observer, reviews each line of code as it is typed in. The two switch roles frequently. Whereas our human-human data collection was not formally set up as pair programming, informally we found that a majority of students did collaborate successfully on the problems we developed. Only two of the fifteen pairs were not successful collaborators; one pair worked in parallel (both had high pre-test scores) and one pair worked collaboratively but got stuck (both had low pre-test scores). Interestingly, while the face-to-face dialogues we collected still displayed some discussions that were grounded in programming experience (i.e., comments about improving the programs they were debugging), this happened markedly less frequently in the computer-mediated interactions. This could be due to changes we made in the problems used for the computer-mediated interactions as a result of reviewing the face-to-face interactions, as was discussed above.

Knowledge Co-construction and Task Initiative

As we discussed earlier, we prefer problem-solving contexts in which the student is actively participating and moving forward with problem solving. Peer interactions is one way in which this may happen. However, peer interactions are not necessarily successful (Barron 2003). One mechanism that has been proposed as underlying those peer interactions that are more beneficial to the participants than others is knowledge co-construction (KCC) (Chan 2001; Hausmann et al. 2004; Mullins et al. 2013; Damşa 2014). A KCC episode is a series of utterances and actions in which students are jointly constructing an understanding or shared meaning of a concept required for problem solving. An example of KCC is shown in Fig. 1. However, KCC is an abstract construct, a "black box" which does not provide insights into what actually happens during learning-conducive peer interactions. Whereas relations such as *elaborate* and *criticize* within KCC episodes have been proposed in order to provide a finer analysis (Hausmann et al. 2004), to recognize such relations, an artificial agent would have to understand the propositions those relations link. This is almost impossible on a general scale given the current state-of-the-art in Artificial Intelligence. Hence we believe that to implement a peer learning agent that is able to collaborate with its interlocutor (the human student) in real time, turn by turn, KCC must be indirectly modelled via shallower and more local constructs. By shallower constructs, we mean ones that require less knowledge and deep reasoning than KCC; by more local constructs, we mean ones that can be recognized by looking at only one or two utterances, not longer stretches of the interaction.

When we analyzed the corpus of human-human peer dialogues that we had collected, we looked for a computationally simpler surrogate for the notion of Knowledge Co-Construction, and we found it in *initiative*. Initiative and initiative shifts emerged from research on collaborative dialogue as a powerful linguistic indicator of participation in dialogue and of contributions to problem-solving activities that take place during dialogue (Walker and Whittaker 1990; Shah et al. 2002; Core et al. 2003; Heeman et al. 2003; Yang and Heeman 2010; Nouri and Traum 2014). A speaker takes initiative when she contributes content which is new with respect to her dialogue partner's prior contributions and is not solicited by her partner. Initiative is often divided into two kinds: *dialogue* initiative and *task* initiative (Jordan and Di Eugenio 1997; Guinn 1998; Chu-Carroll and Brown 1998). *Dialogue* initiative is based on the types of utterances that the interlocutors exchange–that is, assertions, commands, questions, and prompts. *Task* initiative pertains to advancing problem solving. Combining definitions of task initiative from Jordan and Di Eugenio (1997), Guinn (1998), and Chu-Carroll and Brown (1998), we define task initiative as *any action by a participant to achieve a goal directly, decompose a goal, or reformulate a goal*.

If peers are working together to solve a problem, and hence are co-constructing a solution, one would expect *both* peers, not just one, to contribute to the solution. This suggests that successful peer interactions should be rich in *shifts* in initiative (Howard et al. 2015). Indeed we found that task initiative shifts are a promising alternative for recognizing when KCC is taking place (Kersey (Howard) et al. 2008, 2009; Howard et al. 2015). In our data, the frequency of task initiative shifts within human-identified KCC episodes is significantly greater than outside of these episodes, and the frequency of task initiative shifts is significantly correlated with learning (Howard et al. 2015). Thus, to our collaborative problem-solving dialogue agent, we added the ability to track task initiative shifts, so that when the initiative shifts fall below a certain threshold, the agent selects dialogue moves that encourage shifts in task initiative.

Our findings on task initiative shifts and KCC held for problems on linked list data structures, not for stacks, and only weakly for BSTs. An analysis of pretest scores and learning gains by data structure type suggested that our sample of students already had a good understanding of stacks and were not yet prepared enough for binary search tree problems (Howard et al. 2015). Therefore, in our experiments with the agent, we focused on the linked list problems since we targeted students at the same stage as the ones who had participated in our data collection. Hence we expected them to be better prepared to tackle problems on linked lists than on BSTs, but at the same time to still have room to improve their understanding, or to still be in need of the additional practice.

Agent Design

The software agent, KSC-PaL (Kersey (Howard) 2009; Kersey (Howard) et al. 2010), was designed to collaborate with students to solve problems on data structures. The core of KSC-PaL is the TuTalk dialogue toolkit (Jordan et al. 2007). TuTalk supports the creation of natural language dialogue systems for educational applications and allows for both tutorial and conversational dialogues (Jordan 2007; Albacete et al. 2015). To develop KSC-PaL, we added a graphical user interface specifically designed to support solving data structures problems, replaced TuTalk's student model, and augmented its dialogue planner to support the automatic tracking and shifting of task initiative. The focus of our agent description is on the interface and the automatic tracking and shifting of task initiative. We will briefly describe other components of the system to set the stage for the experiment description that follows. Details of the student model and the augmentation of the dialogue planner are described in Kersey (Howard) (2009).

The Interface

A student interacts with KSC-PaL via an interface as shown in Fig. 3 (KSC-PaL is "Amy" in the Log). There are four distinct areas within the interface:

1. **Problem display**: Displays the problem description. (In Fig. 3, the problem description has been abbreviated for this illustration.)
2. **Code display**: Displays the code associated with the problem statement. This section numbers each line of the code for easy reference by the participants. Additionally, the student is able to use a pop-up menu to make changes to the code, such as crossing-out lines and inserting lines, as well as undoing these corrections. However, for the problems we developed and tested for KSC-PaL, no code modifications were requested.
3. **Chat and Log Area**: Accepts student input and provides an interleaved dialogue history of the student-agent participating in the problem solving. The history is logged for later analysis.
4. **Drawing area**[3]: Provides a data structure diagram editor to aid in the explanation of parts of the problem being solved. The diagram editor provides objects representing nodes and links. These objects can then be placed in the drawing area to build lists, stacks, or trees, depending on the type of problem being solved.

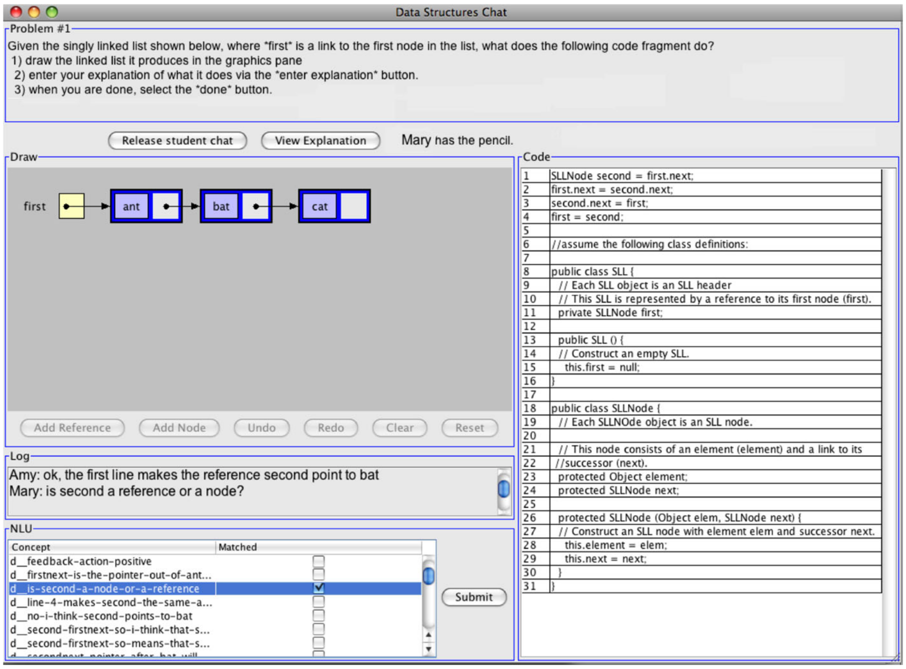
The changes made in the shared workspace (drawing and code areas) are logged and propagated to the partner's window. Further details on the interface can be found in Kersey (Howard) (2009).

Interpreting Students' Actions

Because automatic understanding of unsolicited student responses is time-consuming to develop and the accuracy tends to still be low enough to cause confounds in experiments (e.g., Dzikovska et al. 2014), we did not develop a fully automatic natural language understanding module. Instead we routed the agent's candidate representations for the student's chat input to a human interpreter to review and either accept or override. Thus, the flow of interaction is from student to human interpreter to agent and then back to the student.

The interpreter's interface presents the candidates as a menu and KSC-PaL automatically checks-off what it considers to be the best representation for what the student typed in the chat window (as shown in the NLU area of Fig. 4). The human interpreter reviews the suggested match and either accepts it or selects another better match. If the human interpreter decides that none of the anticipated responses displayed is a good match, then a second list of interpretations, containing student

[3]This area was developed by Davide Fossati for iList (Fossati et al. 2009, 2015).



Problem #1

Given the singly linked list shown below, where "first" is a link to the first node in the list, what does the following code fragment do?
1) draw the linked list it produces in the graphics pane
2) enter your explanation of what it does via the "enter explanation" button.
3) when you are done, select the "done" button.

Mary has the pencil.

Code

```
SLLNode second = first.next;
first.next = second.next;
second.next = first;
first = second;

//assume the following class definitions:

public class SLL {
  // Each SLL object is an SLL header
  // This SLL is represented by a reference to its first node (first).
  private SLLNode first;

  public SLL () {
  // Construct an empty SLL.
    this.first = null;
}

public class SLLNode {
  // Each SLLNOde object is an SLL node.

  // This node consists of an element (element) and a link to its
 //successor (next).
  protected Object element;
  protected SLLNode next;

  protected SLLNode (Object elem, SLLNode next) {
  // Construct an SLL node with element elem and successor next.
    this.element = elem;
    this.next = next;
  }
}
```

Log

```
Amy: ok, the first line makes the reference second point to bat
Mary: is second a reference or a node?
```

NLU

Concept	Matched
d__feedback-action-positive	☐
d__firstnext-is-the-pointer-out-of-ant...	☐
d__is-second-a-node-or-a-reference	✓
d__line-4-makes-second-the-same-a...	☐
d__no-i-think-second-points-to-bat	☐
d__second-firstnext-so-i-think-that-s...	☐
d__second-firstnext-so-means-that-s...	☐

Fig. 4 The Interpreter's interface

initiative utterances, are presented to the interpreter. If none of these match, then all potential concepts that have been defined in TuTalk for KSC-PaL are presented to the interpreter for matching; this last set also includes more general candidates such as, "unanticipated on-topic clarification that is incorrect" and "unanticipated response". The human interpreter is required to find a match within one of the three sets provided. Whereas it may appear that this process is long and involves too many choices for the human, the number of candidates to review does not exceed 30 (the concepts that TuTalk is aware of). Hence, the interpreter has a limited, predetermined set of choices. In this way, his/her intervention is circumscribed. Still, the agent can be slower than a fully automated agent would be. In practice, the wait time ($M = 4.42s, \sigma = 11.62s$) appears to be similar to what the students in the human-human condition experienced, when chatting via the interface, as shown in Fig. 1.

Note that the human interpreter cannot add additional interpretations, or modify the ones offered as candidates by the agent. Once the candidate selection is done, the human interpreter takes a back seat until the next student turn; namely, it is the agent only that computes the response to the student's action/utterance.

Automatic Tracking and Shifting of Task Initiative

We describe the automatic tracking and shifting of task initiative in terms of three tasks. The first *recognizes* the task initiative holder in each utterance or action (task 1); the second *assesses* whether task initiative should shift to the student (task 2); the third *employs* appropriate discourse moves (*shifters*) to encourage task initiative shifts (task 3).

Task 1: Recognizing the Task Initiative Holder

To recognize whether the student or the simulated peer has task initiative, we use machine learning classifiers. The classifiers are trained by providing them with data points consisting of sets of possibly relevant features and the corresponding class. For us, data points are individual utterances and drawing actions. The features are as detailed below, and the classes are derived from manual task initiative annotations which we will describe briefly next. Note that the drawing actions we are calculating the initiative holder for are what we call *interpreted* drawing actions; these are short sequences of atomic drawing actions that together represent an identifiable step in the problem solving process. For example, drawing an arrow is not conceptually meaningful per se; however, drawing an arrow, sizing it and having it point to an existing node means linking a pointer to that node. KSC-PaL automatically recognizes interpreted drawing actions in real time.

We annotated a corpus of 54 human peer interactions (described in more detail in Howard et al. 2015) for task initiative and obtained an interannotator reliability score of $\kappa = 0.68$. The annotators assigned task initiative to a dialogue participant if that participant made an unsolicited:

- suggestion to verify a section of code
- explanation of what a section of code does
- identification of a section of code as correct or incorrect
- suggestion to correct a section of code
- correction to a section of code prior to discussing it with the other participant

As is customary in machine learning, we trained various classifiers using a set of automatically identified features to uncover which one gives us the best performance (see Kersey (Howard) 2009 for further details). These experiments showed that we can obtain better recognition rates for the task initiative holder if we model one classifier for utterances and another for drawing/coding modification actions. Specifically, K^*, a clustering algorithm (Cleary and Trigg 1995), performed best for utterances. K^* correctly classified the task initiative holder in 71.77 % of utterances in our corpus. The features used by the K^* classifier are: current utterance length, the dialogue act label of the student's prior utterance,[4] the holder of dialogue initiative (student or KSC-PaL), and the student's knowledge score so far.

The knowledge score is computed on the basis of the student model. The dialogue initiative holder (this is different from task initiative) is automatically computed based on the automatic annotation of dialogue acts and on Walker and Whittaker's rules for dialogue initiative allocation (Walker and Whittaker 1990). The only

[4]The student's prior utterance may be the last utterance in the student's prior turn, if KSC-PaL had a turn between the student's two utterances.

dialogues acts that are automatically annotated are the ones needed to apply Walker and Whittaker's rules: (1) assertion, (2) command, (3) question, and (4) prompt (a prompt is an utterance not expressing propositional content). Dialogue acts capture the role each utterance plays in the dialogue and can be considered as an operationalization of speech acts (Austin 1962; Searle 1965, 1975). Other features such as uncertainty phrases, length of pauses between utterances, prior turn dialogue act, and whether the statement was incorrect were not helpful in this model.

For recognizing the task initiative holder in drawing actions, JRip (Cohen 1995), a rule-based algorithm, gave the best results. JRip correctly classified the task initiative holder in 86.97 % of drawing actions. The only feature used is the student's knowledge score so far.

While the accuracy results for both classifiers are reasonable, when we compute intercoder agreement scores with respect to the human annotated corpus, κ scores are low: $\kappa = .357$ for recognizing the task initiative holder in utterances, and $\kappa = .465$ in drawing actions. These scores indicate that there is still considerable room for improvement in automatically recognizing the task initiative holder. A next step may be to explore additional features that characterize lexical differences between turns under the assumption that contributing something new to problem solving would introduce some novel lexical items relative to the turn of the previous task initiative holder (consider P3 and P4 in Fig. 1). In contrast, a counter-proposal may be lexically close to the previous task initiative holder's turn (consider P4 and P5 in Fig. 1).

Task 2: Deciding to Shift Initiative

Once the agent has determined who is holding task initiative, it needs to assess whether it is appropriate to encourage a shift in task initiative. To aid in this decision-making, the agent keeps track of the level of task initiative shifts via two measures: 1) the average number of task initiative shifts over all dialogue turns, and 2) the average number of shifts in the last ten dialogue turns. For both measures, dialogue turns include both utterances and drawing actions. These computations are done in real time, and hence represent the average level of task initiative shifts based on what has transpired in the dialogue so far. The agent will encourage task initiative shifts when the smaller of the two measures of automatically recognized task initiative shifts falls below a threshold of 0.2159, and if the student's knowledge score, according to the student model, has not increased again since the last time the agent made a move to promote a task initiative shift. The threshold value was calculated from corpus data and is the average number of task initiative shifts per KCC episode, occurring in list problems in the human peer interactions. The additional condition concerning improvement in the student's knowledge score is due to our attempt to encourage initiative shifts <u>in order to</u> increase learning.

Task 3: Discourse Moves that Engender Task Initiative Shifts

When KSC-PaL decides that task initiative should shift to the human peer, it needs to generate appropriate discourse moves to invite the peer to take initiative. We further analyzed our human-human corpus in view of the literature that identifies discourse moves for task initiative shifts (Walker and Whittaker 1990; Chu-Carroll and Brown 1998; Shah et al. 2002; Bhatt et al. 2004). Those moves are (see their respective frequencies in Table 1):

Hedge: A hedge is a mitigating device used to lessen the impact of an utterance, such as *could* in the sentence *That could be wrong*. The function of hedges has been studied in an educational context–for example, by Evens and her students–and shown to correlate with students' uncertainty (Shah et al. 2002; Bhatt et al. 2004). In turn, the perception of the partner's uncertainty is an indication to the peer that they should take initiative.

Request feedback: When uncertain of his/her contribution, a student may request an evaluation from their peer.

Issue invalid statement: An utterance that expresses incorrect propositional content

Give-up task: An utterance that indicates the contributor is giving up on the task

Prompt: An utterance that does not express propositional content (e.g., *ok, yeah*).

Pause: A silence of 60 s or more after an utterance or action, computed automatically

Given the availability of the moves to invite a partner to take initiative and our annotation for task initiative that we discussed earlier, we then explored the following questions relative to our corpus of human-human dialogues:

1. How often does task initiative shift to the partner, after one of these moves is used? We found that 283 shifts in task initiative or approximately 67 % of all task initiative shifts occurred in response to one of these moves.[5] The remaining shifts were likely an explicit takeover of initiative without preceding moves that encourage shifts.
2. Conversely, how effective is each type of move to engender a shift in task initiative? Table 1 shows the number and percentage of instances of each move that resulted in a task initiative shift. The most likely move to lead to a task initiative shift was an invalid statement.

Hence we chose to have the agent encourage shifts in task initiative when it is the initiative holder by:

- making mistakes which will ideally lead to a student's criticism [We will discuss in the next section whether making mistakes on purpose can engender confusion in the student (Lehman et al. 2012).]
- using prompts–that is, utterances that have no propositional content, such as *"yes?" "hmmm"*
- hedging–that is, showing uncertainty by prefacing one's statements with expressions such as *I think*
- requesting feedback, via expressions such as *What do you think?*

[5]The number of task initiative shifts due to these moves is lower than just summing up the numbers in the second column of Table 1. This is because more than one move may appear in the same utterance; for example, a pause followed by a hedge.

Table 1 "Shifters": discourse moves to shift task initiative

Discourse move	Total number	Number (Percent) of instances that led to task initiative shift
Invalid statement	132	51 (38.64 %)
Prompt	560	164 (29.29 %)
Pause	182	46 (25.27 %)
Hedge	142	34 (23.94 %)
Request feedback	233	51 (21.88 %)
Give-up task	20	4 (20.00 %)

As concerns pauses, our human-human data confirms they are indicative of some initiative shifts. However, we decided to exclude them from the agent's repertoire since a pause by the agent might be misinterpreted by the human peer as a system error. We also excluded giving up the task, since there were so few instances of this move in our data to start with.

Figure 5 shows an agent-peer interaction in which KSC-PaL makes a move in line S1 to shift initiative by contributing an invalid statement. In S2, the human peer sees the error and corrects it, thereby contributing new content that was not invited. In contrast, Fig. 6 shows KSC-PaL making a move in line F4 to shift initiative by requesting feedback. In F5, the human just acknowledges understanding and does not contribute new content. Thus, KSC-PaL retains task initiative. In both dialogue excerpts, KSC-PaL is the initiative holder at the beginning. At the end of the dialogue in Fig. 5, the human is the initiative holder whereas in Fig. 6, KSC-PaL is the initiative holder throughout.

If instead the human is the initiative holder when KSC-PaL decides task initiative should shift, it chooses a correct response relative to the student's last contribution as this is a move that is more likely to shift initiative back to the agent itself. The correct response KSC-PaL has available at a particular point in the dialogue attempts to reflect a typical correct response that a human student gave in the past (in our human-human dialogues); hence it may or may not result in the current student relinquishing initiative.

Evaluating Automated Attempts to Shift Initiative

We turn now to an experiment in which CS students interacted with one of two versions of the KSC-PaL agent; one that tracked task initiative shifts and encouraged them when needed, as just described, (Experimental) and one that did not track task

```
S1]   Agent: ok, the first line makes the reference second point to ant
S2]   Human: are you sure about that?  i blieve it points to bat
S3]   Agent: oh yeah, that's right
```

Fig. 5 Agent-to-Peer interaction with shifter that succeeds

```
F1]   Agent: lines 6-9 are for case pred !=null
F2]   Human: what is pred?
F3]   Agent: i think that pred is the node that the code inserts
             the new code after
F4]   Agent: so, what do you think?
F5]   Human: okay it clear to me now
```

Fig. 6 Agent-to-Peer interaction with shifter that fails

initiative shifts and thus could not intentionally encourage them (Control). The experiment explores whether students learn when interacting with the agent, whether there are differences in student learning gains depending on the agent version with which they interacted, and whether the Experimental version of the agent is able to influence task initiative shifts relative to the Control version of the agent.

Participants

Twenty-six students were recruited from introductory computer science courses at the University of Pittsburgh, the University of Illinois at Chicago, DePaul University and Carnegie Mellon University. Thirteen students (twelve males, one female) interacted with the experimental agent and thirteen (eleven males, two females) interacted with the control agent. One male subject in the control condition was excluded as an outlier from all analyses (with one exception, as will be noted). The subject had taken the data structures course six years earlier and his problem-solving time was above average (48 min vs. 29 min).

We did not formally assess the participants' English proficiency, which could affect their willingness to chat. Our informal observations based on our interactions with the participants indicate that all participants were native or at least fluent speakers – no subject had noticeable difficulty with English. (Two of the four authors were in charge of each experiment run at the two sites; namely they were present and directly interacted with the subjects in all portions of the experiment.)

Materials

Each student was presented with two problems on linked list data structures. Both problems were code explanation, as described earlier.

The pretest and post-test were identical (see Appendix). It was a five-item test on linked lists; three items were from the test used in our human-human data collection and two new items were added. All questions in the test were carefully crafted to assess a deep level of knowledge of the data structures of interest. They were based on the considerable experience of two authors of this paper and of collaborators in a tutoring project in teaching precisely these data structures. Additionally, the specific problems were modelled on problems from textbooks and/or from previous exams administered by those among us who had taught those topics. Hence we consider these tests valid measures of knowledge on linked lists.

Procedure

Each student was given 25 min. to take the five-item pre-test to measure his/her knowledge of linked list data structures prior to the collaborative interaction with the agent. This was followed by a short tutorial on using the interface (10 min). The student was then informed that he/she would be using the interface to collaborate with an artificial agent to solve CS data structures problems. The student was told that he/she could chat as one normally would with a fellow student, cautioned that the agent would not always be correct and told that because the agent's language understanding capability was limited, that it would get an assist from an experimenter who would help it interpret the student's actions and chat but that the experimenter would not help the agent in any other way. We also cautioned the student that the agent might be slow at times because of its human helper.

The students were randomly assigned to work with either the experimental or control version of KSC-PaL. The students then engaged in the collaborative problem-solving activity with the agent using the interface. The interaction with the agent was for approximately 30 minutes ($M = 29.29$, $\sigma = 10.92$). The session concluded with each student being given 20 min to complete a post-test (identical to the pre-test).

Results

Table 2 shows pre- and post-test scores and learning gains, by condition, and cumulatively across conditions. To investigate whether students learned from pre-test to post-test, a one-way within-subjects (repeated measures) ANOVA was conducted. There was a significant difference in learning for all students combined and for the experimental students as shown in Table 2.

To investigate the effect of condition (Control vs. Experimental) on students' pre to post-test scores, a one-way within-subjects (repeated measures) ANOVA was conducted. There was a significant effect of using the system on test scores (Wilks' $\lambda = .74$, $F(1, 23) = 8.01$, $p = .01$). However, there was no significant effect of condition on test scores (Wilks' $\lambda = .99$, $F(1, 23) = .02$, $p = .90$). These results indicate that a collaborative problem-solving agent such as KSC-PaL is able to help students improve their knowledge of linked lists even if the impact on learning of tracking initiative and attempting to change the student's collaborative behavior was similar to not doing so.

Table 2 Student learning using KSC-PaL

Condition	N	Pre-test M	Post-test M	Gain	*F*	*p*
Control	12	0.643 ($\sigma = 0.166$)	0.697 ($\sigma = 0.160$)	0.053 ($\sigma = 0.114$)	$F(1, 11) = 2.6$	$< .14$
Experimental	13	0.603 ($\sigma = 0.190$)	0.661 ($\sigma = .0179$)	0.058 ($\sigma = 0.083$)	$F(1, 12) = 6.5$	$< .03$
All students	25	0.622 ($\sigma = 0.176$)	0.678 ($\sigma = 0.167$)	0.056 ($\sigma = 0.097$)	$F(1, 24) = 8.4$	$< .01$

We turn now to the question of whether the agent succeeded in encouraging task initiative shifts. First, we examine whether it was effective in generating more task initiative *shifters*, namely moves that are conducive to shifting task initiative. Tables 3 and 4 show the number of utterances produced by KSC-PaL which did not contain / contained shifters, by condition over all interactions (Table 3) and per student interaction (Table 4). Tables 3 and 4 also show a lower bound estimate[6] of how many shifters actually caused the student to take initiative (*uptake*).

The reader may be confused in that shifters were produced in the control condition as well. As we noted, some shifters such as *prompts* are common linguistic patterns that may arise for other reasons; for example, because the agent does not have anything new to contribute at this point in the conversation. Table 3 shows that more shifters were generated in the experimental condition; the difference is statistically significant ($\chi^2 = 24.2, p = 0.000$). This result is confirmed by additional analysis. A one-way between subjects ANOVA was conducted to compare the effect of KSC-PaL condition on number of shifters. There was a significant effect of condition on shifters ($F(1, 23) = 10.57, p < .01$). A one-way between subjects ANOVA was conducted to compare the effect of KSC-PaL condition on uptake. There was also a significant effect of condition on uptake ($F(1, 23) = 7.64, p = .01$). A multiple regression was conducted to see if condition and frequency of shifters predicted the amount of initiative uptake during the dialogues. Using the Enter method, it was found that condition and frequency of shifters explain a significant amount of variance in amount of uptake of initiative ($F(2, 22) = 54.58, p < .01, R^2 = .83, R^2 Adjusted = .82$). Furthermore, the analysis shows that condition did not significantly predict amount of uptake ($\beta = -.02, t(22) = -.17, p < .87$); however, frequency of shifters did ($\beta = .92, t(22) = 8.74, p < .01$). So, condition predicts shifters but not uptake. Instead, uptake is predicted by shifters when condition is factored out.

Before moving on to our last question of whether task initiative shifts in interactions with an automated peer correlate with learning as they do in human-human peer interactions, we will look more closely at the performance of the three individual initiative shifting tasks. We evaluated Task 1, recognizing who holds task initiative, on a larger number of items by manually annotating task initiative for all 937 student utterances and drawing actions processed by KSC-PaL across the two conditions (not just the experimental condition, where Task 1 was used in real time)[7]. This data included the one outlier subject in the control condition who was excluded from all other analyses. The level of accuracy of Task 1 was as expected: 80.15 % (i.e., 747) of the 937 utterances and drawing actions were correctly classified.

[6]It is an estimate because it is calculated by finding annotated agent shifters and checking the next turn to see if initiative shifted. Sometimes the next turn is another agent turn. So this method of counting may underestimate uptake.

[7]We did not measure intercoder agreement when we annotated this corpus because it is reasonable to assume it would be the same as it was when we measured it for the human-human corpus: for the human-human corpus, $\kappa = .68$ for task initiative annotations (Howard et al. 2015). This is high enough to support tentative conclusions and was comparable to agreement measures on other corpora for similar definitions of task initiative (Howard et al. 2015).

Table 3 Generation and uptake of initiative shifting moves across both versions of KSC-PaL (all interactions)

	Non-shifters	Shifters	Uptake
Control	193	33	18
Experimental	155	81	47

As concerns Task 2, deciding to shift initiative, even if this task was effective in generating more initiative shifters than in the control condition, as shown in Table 3, in retrospect, we believe the experimental condition generated too few potential initiative shifters. The threshold we set for the average level of task initiative shifts was derived from the human data, and it is equivalent to one task initiative shift every five utterances, on average. However, the human dialogues are three times longer, 55 utterances on average as opposed to only 18 utterances on average in the human-computer dialogues (to be precise, $M = 55.00, \sigma = 34.83$ in the human-human dialogues and $M = 17.82, \sigma = 7.60$ in the human-computer dialogues). This means that in the human-computer dialogues, there were only two or three opportunities to induce a task initiative shift. (Since KSC-PaL begins the dialogue, it cannot attempt to shift initiative until utterance 3 at the earliest; that is, after the student has had at least one opportunity to respond).

As concerns Task 3, generating discourse moves that encourage task initiative shifts, 31 hedges, 21 requests for feedback, 18 prompts, and 11 incorrect statements were produced. Whereas this distribution does not reflect the relative frequencies of discourse moves in Table 1, we remind the reader that a linguistic form that invites the student to take task initiative may be generated for a different reason (e.g., the system is not *able* to suggest a next move vs. chooses not to suggest a next move and therefore generates a prompt). On the other hand, the only reason to generate an incorrect statement is to try to get the student to take task initiative. Producing mistakes on purpose may raise concerns that the student may get confused. For the eleven incorrect statements produced in the experimental condition, six times the student corrected the agent, and once the student at first seemed frustrated ("ok, why don't you try?") and then agreed with the agent. The agent then "changed its mind". The remaining four times were ignored by the student at first but then three of them were addressed when the agent repeated the incorrect statement; only one was completely ignored. So although the student may be confused at least initially by the incorrect statement, the student is usually able to correct the error. Instead of always being counterproductive to learning, a state of confusion can create opportunities for learning if managed properly (Lehman et al. 2012).

Finally, we used multiple linear regression with pre-test score and task initiative shifts as a covariate to test for correlations between student learning and task initiative shifts when working with an artificial agent. As we mentioned, in the control condition KSC-PaL does not encourage task initiative shifts; however, it doesn't prevent those either. Hence, in order to evaluate the effectiveness of initiative shifts, the following analysis uses subjects from both conditions and, specifically, the student

Table 4 Generation and uptake of initiative shifting moves across both versions of KSC-PaL (per student interaction)

	Non-shifters	Shifters	Uptake
Control	M= 16.08 ($\sigma = 3.66$)	M= 2.75 ($\sigma = 2.09$)	M= 1.50 ($\sigma = 1.31$)
Experimental	M= 12.77 ($\sigma = 4.11$)	M= 6.23 ($\sigma = 3.11$)	M= 3.62 ($\sigma = 2.33$)

sentences and actions that were manually annotated with respect to initiative shifts, as mentioned earlier for Task 1, but minus the outlier subject.

We used two measures of task initiative shifts to run two independent multiple regressions: (1) the number of task initiative shifts ($M = 11.76, \sigma = 4.53$) and [8] (2) the number of normalized task initiative shifts, calculated by dividing the number of task initiative shifts by the total number of utterances and drawing actions for the session. Hence, Table 5 shows the results of the two independent multiple regressions, after regressing out the impact of pre-test score. Table 5 shows that while task initiative shifts are significant or marginally significant predictors of post-test score, the impact is relatively small since the variance in task initiative explains little of the variance in the post-test (i.e., the change in R^2 is small).

If the same multiple linear regression analysis, where pre-test score and task initiative shifts are covariates, is applied to those subjects with a pre-test score below the mean (see Table 6), there is a larger impact of task initiative shifts on post-test score but it is not significant. Two additional multiple regression analyses were run for all subjects and low-pretest subjects; along with pre-test score and normalized initiative shifts, they included condition as a covariate. For all subjects, condition was not found to be significantly correlated to post-test score ($\beta = 0.42$, $t(24) = 0.384$, $p = .705$). Likewise, analysis of low pre-test subjects showed no significant correlation of post-test score with condition.

These results differ from those for human-human interactions (see Howard et al. 2015) in that frequency of task initiative shifts was a significant predictor of post-test score for low pre-test subjects while only marginally significant in the human-computer interactions. Additionally, the impact of task initiative shifts for human-human interactions is much higher (explains more variance) than in the human-computer interactions.

Discussion

With our creation of the artificial agent, KSC-PaL, we have sought to inform educational technology that can support CS education, specifically at the beginning of students' careers, when they are more prone to stumble on the foundations of CS knowledge. We showed that a peer collaborative agent for education is feasible and

[8] $M = 11.76$ may seem too high, since the dialogues contain only 18 utterances on average. However, recall that task initiative shifts are computed with respect to utterances **and** drawing actions. There are on average 36 total utterances and drawing actions per dialogue ($M = 35.85$, $\sigma = 10.56$).

Table 5 Impact of task initiative shifts on learning, in KSC-PaL (all subjects)

Predictor of post-test	R^2	β	t, p for predictor
Pre-test (n= 25)	0.710	0.843	$t = 7.503$, $p = 0$
Pre-test +	0.739	0.900	$t = 7.817$, $p = 0$
Task initiative shifts		0.179	$t = 1.551$, $p = .135$
Pre-test +	0.761	0.896	$t = 7.503$, $p = 0$
Normalized task initiative shifts		0.233	$t = 2.179$, $p = .04$

that students' knowledge of linked lists improved after interacting with it. Data structures often present difficulties for students, as CS educators have noted (AA.VV. 2001, 2007, 2013; Scime 2008). As we discussed in the Introduction, linked lists are the first *recursive* data structures CS majors encounter, and recursion is cognitively difficult to grasp, whether in CS (Pirolli and Anderson 1985; Pirolli and Recker 1994; Wing 2006), or in natural language (Miller and Chomsky 1963; Anderson 1976; Karlsson 2007). A second reason we hypothesize data structures are difficult for students is that, as in physics or chemistry (Kozma 2003; Meltzer 2005), students must be able to negotiate multiple presentations of the same concept. KSC-PaL is able to collaborate with students using those same representations and is unique in CS education, and education in general, because it engages a student in a one-on-one collaborative problem-solving dialogue while treating the student as a peer. The peer status of the agent seems to encourage students to look more critically at the agent's contributions. On the other hand, the agent does not leave the student to flounder on his/her own for long when problem solving stalls. Because students were able to learn while interacting with KSC-PaL, peer collaborative agents could serve as an alternative to or supplement for ITSs.

Earlier in the paper we touched on our concerns about underrepresentation and retention of females and minorities in CS. We hypothesize that KSC-PaL's interface could promote the seamless integration of multiple CS representations and that collaborative problem solving that focuses on the semantics rather than the syntax of the code could be of particular value in this regard: it could help mitigate attitudes that are expected to be detrimental to learning and seem to be particularly off-putting to female students. However, we must leave it for future work to test these hypotheses.

Table 6 Impact of task initiative shifts on learning, in KSC-PaL (low pre-test subjects)

Predictor of post-test	R^2	β	t, p for predictor
Pre-test (n= 13)	0.391	0.626	$t = 2.659$, $p < .03$
Pre-test +	0.501	0.731	$t = 3.129$, $p < .02$
Task initiative shifts		0.350	$t = 1.486$, $p < .20$
Pre-test +	0.554	0.684	$t = 3.205$, $p < .01$
Normalized task initiative shifts		0.408	$t = 1.914$, $p < .09$

To create a collaborative peer agent for educational purposes, we needed to create not just a collaborative problem-solving agent but one that would be of educational value to the student. Thus, we endeavored to embed within KSC-PaL the ability to monitor the student's collaborative behavior and then make discourse moves that could help alter the student's behavior, if necessary, while still maintaining its role as peer. Work on collaboration with peers suggests that knowledge co-construction (KCC), which is a series of utterances and actions in which students jointly construct an understanding or shared meaning of a concept required for problem solving, is a mechanism that is beneficial to *both* participants when it occurs. KCC also echoes the constructivist view that participation is important to learning (Resnick 1989; Ben-Ari 1998). However, building an artificial agent based on our current understanding of KCC is infeasible. Thus we turned to a surrogate for KCC, task initiative shifts. Monitoring task initiative shifts is appealing because it is more feasible to implement and because task initiative represents a new contribution towards the construction of a solution. Furthermore, task initiative shifts could serve as a measure of the degree to which the construction of the solution is a joint activity since there is a correlation between the frequency of task initiative shifts and learning (Howard et al. 2015). While initiative and *dialogue* initiative shifts (who is contributing something new to the dialogue) have been studied in the context of tutorial interactions (Core et al. 2003; Shah et al. 2002; Bhatt et al. 2004), *task* initiative shifts have not previously been studied or applied in any educational context.

The KSC-PaL agent shows that it is feasible to recognize task initiative and task initiative shifts, and to take actions that encourage a shift in task initiative to the peer student, automatically and in real time. We developed and described three tasks for encouraging the student to take task initiative: 1) recognize the task initiative holder in the current utterance or action; 2) assess whether initiative should shift; 3) generate discourse moves that encourage task initiative shifts. We showed that the agent achieved an acceptable level of performance on each of the three tasks, although there is room for improvement in recognizing the task initiative holder and deciding when to attempt to shift the initiative. Additional automatically recognizable features that reflect the task content being presented may help to further improve recognition of the task initiative holder. Lowering the threshold for when initiative should be shifted is advisable given that KSC-PaL's interactions with students are considerably shorter than the interactions between human peers. Most importantly, we showed that the experimental version of KSC-PaL does generate significantly more shifting moves and shifts than the control version does (which does not track or attempt to adjust task initiative shifts). This indicates that KSC-PaL was able to detect times when the student was letting it do more of the problem solving and that its tactics for encouraging the student to begin taking the initiative again were helpful. Task initiative shifts in the human-artificial peer dialogues do still correlate with learning, but follow different patterns from those found in Howard et al. (2015) for human peers. In addition, correlations in the human-computer dialogues are not as strong as those in the human-human dialogues, which suggests there is still room for improvement in the agent.

Much work remains to be done to improve KSC-PaL and make it a stand-alone agent that does not need the help of a human interpreter and to improve its dialogue capabilities. Currently, it is similar to a hybrid of a system-only initiative agent when it has something it wants to contribute and a "stimulus-response" agent when the student wants to contribute something new. By this we mean that when the agent initiates, it has a plan it is carrying out. In contrast, when the student initiates, the agent only reacts in a shallow way to the student's initiative and does not recognize the student's underlying problem-solving intentions. While it can appear to build upon a student's plan with its current turn, it may not continue contributing to it over multiple turns. Thus the ability to jointly construct a solution is less than ideal.

As concerns scaling up by removing the human interpreter, we have yet to analyze the number of times the human interpreter overrode the agent's interpretation of the student's contribution. Assuming its performance would be consistent with that in Dzikovska et al. (2014), and the improvements we described were made to the agent's initiative-handling ability, it is possible that students would react less negatively to misinterpretations in a peer-to-peer interaction than in a tutorial interaction (i.e., rapport may not be damaged) because they do not always expect their peers to know what to do next. So the peer agent's follow-up after misunderstanding or failing to understand could be interpreted as an initiative or the agent not knowing what to contribute. In addition, the human interpreter also had other general candidates beyond "unanticipated", for which different automated recognition approaches may be more reliable.

An alternative way to scale up would be to use components of the agent within other applications. For example, the initiative-tracking and shifting algorithms could track interactions between two human peers, suggest to one or the other to give their partner a chance to take initiative, and suggest ways in which it could be done.

Finally, while students who interact with the peer collaborative agent KSC-PaL do learn, there was no difference in learning gains due to tracking and influencing initiative shifts vs. not doing so. There are several possible reasons why there was no measurable difference. First, the effect may be too small to determine a significant difference given our small population sample. At the time we ran the experiment, enrollments in CS were at a record low. With today's increased enrollments, recruiting a larger number of students would be less of a challenge. Second, the performance of the agent has much room for improvement and unusual behaviors could dampen learning. Finally, since it is reasonable to expect differences due to dialogue behaviors to have relatively small effects on learning, it may be advisable to use a weaker control agent such as one that actively works to avoid task initiative shifts.

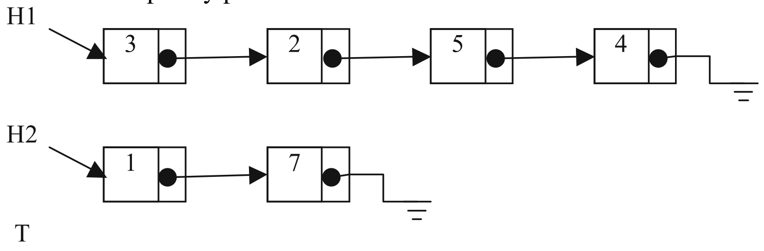
While students do learn during peer collaborations, we leave it to future work to determine whether this style of interaction can be more beneficial than tutorial styles of interaction for certain groups of learners (e.g., high vs. low knowledge students), or whether peer interactions might instead offer an alternative to tutorial interactions or different benefits.

Acknowledgments We thank our anonymous reviewers for their helpful, detailed feedback and Patricia Albacete for advice on analyzing the results. For financial support we gratefully acknowledge the National Science Foundation (ALT-0536968 and ALT-0536959) and the Office of Naval Research (N000140010640).

Appendix: Pre- and Post-Test

Data Structures Quick Test

You have the following two **linked lists**, starting from the head pointers H1 and H2. You also have a temporary pointer T.



1. Look at the following procedure. The procedure is written in pseudo C/C++/Java, but don't worry about programming details such as declarations etc. What is the status of the data structures after its execution? Draw a picture representing them.

```
T = H2;
while (T.next ≠ null) {
     T = T.next;
}
T.next = H1;
```

2. Consider the following "variation" of the same procedure. Why doesn't it work?

```
T = H2;
while (T ≠ null) {
     T = T.next;
}
T.next = H1;
```

3. Look at the following code fragment for deleting an input node `del` from a singly-linked list headed by `first`. It is written in Java but don't worry about programming details. What are the two possible singly-linked list configurations for which it will it not work?

```
public void delete (SLLNode del) {
  // Delete node del in this SLL.
  SLLNode next = del.next;
  SLLNode pred = first;
  while (pred.next != del)
     pred = pred.next;
  pred.next = next;
}
```

You may assume the following class definitions:

```
public class SLL {
  // Each SLL object is an SLL header.
  //  This  SLL  is  represented  by  a  reference  to  its  first  node
(first).
  private SLLNode first;

  public SLL () {
  //Construct an empty SLL.
    this.first = null;
  }

public class SLLNode {
  // Each SLLNode object is an SLL node.
  // This node consists of an element (element) and a link to its
  // successor (next).
  protected Object element;
  protected SLLNode next;

  protected SLLNode (Object elem, SLLNode next) {
  // Construct an SLL node with element elem and successor next.
    this.element = elem;
    this.next = next;
  }
}
```

4. Draw the list created by the following code. Assume the definition of SLLNode from problem 3.

```
        SLLNode list = new SLLNode(10, null);
        SLLNode list1 = new SLLNode(14, null);
        list.next = list1;
        list1 = new SLLNode(18, null);
        list1.next = list.next;
        list.next=list1;
```

5. A doubly-linked list contains nodes that have links to both the succeeding and preceding nodes. The following code is used with the doubly-linked list class DLL. Assume it is similar to the SLL class defined above except that each node has two references, next and previous. Explain what each line does and the end result of the execution of the method.

```
public void mystery(DLLNode node1, DLLNode node2){
     node1.next = node2.next;
     node2.next.previous = node1;
     node1.previous = node2;
     node2.next = node1;

 }
```

References

AA. VV. (2001). Computing Curricula 2001 – Computer Science. Association for Computing Machinery, and IEEE Computer Society. Report of the Joint Task Force.

AA. VV. (2007). Subject benchmark statement: computing. The quality assurance agency for higher education, United Kingdom.

AA. VV. (2013). Computer science curriculum 2013. Association for Computing Machinery, and IEEE Computer Society. Report of the Joint Task Force.

AA. VV. (2014). Occupational outlook handbook. Technical report, US Department of Labor. http://www.bls.gov/ooh.

Albacete, P., Jordan, P.W., & Katz, S. (2015). Is a dialogue-based tutoring system that emulates helpful co-constructed relations during human tutoring effective? In *AIED 2015, the 17th international conference on artificial intelligence in education*.

Alzoubi, O., Fossati, D., Di Eugenio, B., & Green, N. (2014). ChiQat-Tutor: an integrated environment for learning recursion. In *Proceedings of the 2nd workshop on AI-supported education for computer science (AIEDCS) (at ITS 2014)*, Honolulu, HI.

Anderson, J.R. (1976). *Language, memory, and thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Austin, J.L. (1962). *How to do things with words*. Oxford: Oxford University Press.

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.

Barron, B. (2003). When smart groups fail. *Journal of the Learning Sciences*, *12*(3), 307–359.

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bull.*, *37*(2), 103–106.

Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE Bull.*, *30*(1), 257–261.

Beyer, S. (2014). Why are women underrepresented in computer science? Gender differences in stereotypes, self-efficacy, values, and interests and predictors of future CS course-taking and grades. *Computer Science Education*, *24*(2–3), 153–192.

Bhatt, K., Evens, M., & Argamon, S. (2004). Hedged responses and expressions of affect in human/human and human computer tutorial interactions. In *Proceedings of the 26th annual meeting of the cognitive science society*.

Blackwell, L.S., Trzesniewski, K.H., & Dweck, C.S. (2007). Implicit theories of intelligence predict achievement across an adolescent transition: a longitudinal study and an intervention. *Child Development*, *78*(1), 246–263.

Boyer, K., Phillips, R., Ingram, A., Ha, E., Wallis, M., Vouk, M., & Lester, J. (2011). Investigating the relationship between dialogue structure and tutoring effectiveness: a hidden markov modeling approach. *International Journal of Artificial Intelligence in Education*, *21*(1), 65–81.

Brienza, V. (2012). Jobsrated.com: the 10 best jobs of 2012. http://www.careercast.com/jobs-rated/10-best-jobs-2012.

Cai, Z., Feng, S., Baer, W., & Graesser, A. (2014). Instructional strategies in trialogue-based intelligent tutoring systems. In R.A. Sottilare, A. Graesser, X. Hu, & B. Goldberg (Eds.), *Design recommendations for intelligent tutoring systems: volume 2 - instructional management* (pp. 225–236). U.S. Army Research Laboratory.

Chan, C. (2001). Peer collaboration and discourse patterns in learning from incompatible information. *Instructional Science*, *29*(6), 443–479.

Chi, M.T.H. (2009). Active-constructive-interactive: a conceptual framework for differentiating learning activities. *Topics in Cognitive Science*, *1*, 73–105.

Chu-Carroll, J., & Brown, M.K. (1998). An evidential model for tracking initiative in collaborative dialogue interactions. *User Modeling and User-Adapted Interaction*, *8*(3–4), 215–253.

Cleary, J.G., & Trigg, L.E. (1995). K*: an instance-based learner using an entropic distance measure. In *Proceedings of the 12th international conference on machine learning* (pp. 108–114).

Cohen, W.W. (1995). Fast effective rule induction. In *Proceedings of the 12th international conference on machine learning* (pp. 115–123).

Constantino-González, M., Suthers, D., & Escamilla De los Santos, J. (2003). Coaching web-based collaborative learning based on problem solution differences and participation. *International Journal of Artificial Intelligence in Education*, *13*(2–4), 263–299.

Corbett, A.T., & Anderson, J.R. (1990). The effect of feedback control on learning to program with the LISP tutor. In *Proceedings of the 12th annual conference of the cognitive science society* (pp. 796–803).

Core, M.G., Moore, J.D., & Zinn, C. (2003). The role of initiative in tutorial dialogue. In *EACL '03: Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 67–74). Morristown, NJ: Association for Computational Linguistics.

Damşa, C.I. (2014). The multi-layered nature of small-group learning: productive interactions in object-oriented collaboration. *International Journal of Computer-Supported Collaborative Learning* (pp. 1–35).

DeClue, T., Kimball, J., Lu, B., & Cain, J. (2011). Five focused strategies for increasing retention in computer science 1. *Journal of Computing Sciences in Colleges*, *26*(5), 252–258.

Di Eugenio, B., Fossati, D., Ohlsson, S., & Cosejo, D. (2009). Towards explaining effective tutorial dialogues. In *Proceedings of CogSci 2009, the annual meeting of the cognitive science society*. Amsterdam, The Netherlands.

Dzikovska, M., Steinhauser, N., Farrow, E., Moore, J., & Campbell, G. (2014). BEETLE II: deep natural language understanding and automatic feedback generation for intelligent tutoring in basic electricity and electronics. *International Journal of Artificial Intelligence in Education*, *24*(3), 284–332.

Ezen-Can, A., & Boyer, K. (2013). In-context evaluation of unsupervised dialogue act models for tutorial dialogue. In *Proceedings of the SIGDIAL 2013 conference* (pp. 324–328). Metz: Association for Computational Linguistics.

Ezen-Can, A., & Boyer, K.E. (2015). A tutorial dialogue system for real-time evaluation of unsupervised dialogue act classifiers: exploring system outcomes. In *Proceedings of the international conference on artificial intelligence in education*.

Falkner, N., & Falkner, K. (2012). A fast measure for identifying at-risk students in computer science. In *Proceedings of the 9th annual international conference on international computing education research*(pp. 55–62). ACM.

Fisher, A., Margolis, J., & Miller, F. (1997). Undergraduate women in computer science: experience, motivation and culture. *ACM SIGCSE Bulletin*, *29*(1), 106–110.

Fossati, D., Di Eugenio, B., Brown, C., Ohlsson, S., Cosejo, D., & Chen, L. (2009). Supporting Computer Science curriculum: exploring and learning linked lists with iList. *IEEE Transactions on Learning Technologies, Special Issue on Real-World Applications of Intelligent Tutoring Systems*, *2*(2), 107–120.

Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., & Chen, L. (2015). Data driven automatic feedback generation in the iList intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, *10*(1), 5–26.

Fuller, U., Johnson, C.G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., Lahtinen, E., Lewis, T.L., Thompson, D.M., Riedesel, C., & Thompson, E. (2007). Developing a computer science-specific learning taxonomy. *SIGCSE Bull.*, *39*(4), 152–170.

Graesser, A.C., Lu, S., Jackson, G.T., Mitchell, H.H., Ventura, M., Olney, A., & Louwerse, M.M. (2004). AutoTutor: a tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers*, *36*(13), 180–192.

Guinn, C.I. (1998). An analysis of initiative selection in collaborative task-oriented discourse. *User Modeling and User-Adapted Interaction*, *8*(3–4), 255–314.

Gürer, D., & Camp, T. (2002). An acm-w literature review on women in computing. *ACM SIGCSE Bulletin*, *34*(2), 121–127.

Harsley, R. (2015). Learning together: expanding the one-to-one its model for computer science education. In *Doctoral consortium at the 11th international conference on computing education research*. Omaha, NE.

Hausmann, R.G., Chi, M.T., & Roy, M. (2004). Learning from collaborative problem solving: an analysis of three hypothesized mechanisms. In K. Forbus, D. Gentner, & T. Regier (Eds.), *Proceedings of the 26th annual conference of the cognitive science society* (pp. 547–552). Mahwah, NJ.

Heeman, P.A., Yang, F., & Strayer, S.E. (2003). Control in task-oriented dialogues. In *EUROSPEECH-2003* (pp. 209–212).

Howard, C., Di Eugenio, B., Jordan, P., & Katz, S. (2015). Using initiative to operationalize knowledge co-construction during collaborative problem solving. *Cognitive Science*. To appear.

Hundhausen, C.D., Douglas, S.A., & Starko, J.T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, *13*(3), 259–290.

Hunt, J.M. (1961). *Intelligence and experience*. Ronald.

Jordan, P. (2007). Topic initiative in a simulated peer dialogue agent. In *AIED 2007, the 13th international conference on artificial intelligence in education* (pp. 581–583).

Jordan, P.W., & Di Eugenio, B. (1997). Control and initiative in collaborative problem solving dialogues. In *Working notes of the AAAI spring symposium on computational models for mixed initiative*(pp. 81–84). Menlo Park, CA.

Jordan, P.W., Hall, B., Ringenberg, M.A., Cue, Y., & Rosé, C.P. (2007). Tools for authoring a dialogue agent that participates in learning studies. In *Artificial intelligence in education, AIED 2007* (pp. 43–50).

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, *43*, 365–392.

Katz, S. (2006). Gendered attrition at the undergraduate level. In E. Trauth (Ed.), *Encyclopedia of gender and information technology* (pp. 714–720). Hershey: Idea Group Publishing.

Katz, S., Allbritton, D., Aronis, J., Wilson, C., & Soffa, M.L. (2006). Gender, achievement, and persistence in an undergraduate computer science program. *SIGMIS Database*, *37*(4), 42–57.

Katz, S., Aronis, J., Allbritton, D., Wilson, C., & Soffa, M. (2003). Gender and race in predicting achievement in computer science. *IEEE Technology and Society Magazine*, *22*(3), 20–27.

Kersey (Howard), C. (2009). Knowledge co-construction and initiative in peer learning interactions. PhD thesis, University of Illinois at Chicago.

Kersey (Howard), C., Di Eugenio, B., Jordan, P., & Katz, S. (2010). KSC-PaL: a peer learning agent. In *ITS 2010, the 10th International conference on intelligent tutoring systems* (pp. 72–81). Pittsburgh: Springer.

Kersey (Howard), C., Di Eugenio, B., Jordan, P., & Katz, S. (2009). Knowledge co-construction and initiative in peer learning interactions. In *AIED 2009, The 14th International Conference on Artificial Intelligence in Education*. Brighton, UK.

Kersey (Howard), C., Di Eugenio, B., Jordan, P.W., & Katz, S. (2008). Modeling knowledge co-construction for peer learning interactions. In *ITS 2008, the 9th international conference on intelligent tutoring systems, student research workshop*. Montreal, Canada.

Kozma, R. (2003). The material features of multiple representations and their cognitive and social affordances for science understanding. *Learning and Instruction*, *13*, 205–226.

Lane, H.C., & VanLehn, K. (2003). Coached program planning: dialogue-based support for novice program design. *SIGCSE Bull.*, *35*(1), 148–152.

Latham, A., Crockett, K., & McLean, D. (2014). An adaptation algorithm for an intelligent natural language tutoring system. *Computers & Education*, *71*, 97–110.

Lehman, B., D'Mello, S., & Graesser, A. (2012). Confusion and complex learning during interactions with computer learning environments. *The Internet and Higher Education*, *15*(3), 184–194.

Li, Z., Plaue, C., & Kraemer, E. (2013). A spirit of camaraderie: the impact of pair programming on retention. In *IEEE 26th conference on software engineering education and training (CSEE&T)* (pp. 209–218). IEEE.

Lister, R., & Leaney, J. (2003). Introductory programming, criterion-referencing, and bloom. In *SIGCSE'03: proceedings of the 34th SIGCSE technical symposium on computer science education* (pp. 143–147). New York: ACM Press.

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. *Computer Science Education*, *18*(2), 67–92.

Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C.S., & Thomas, L. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. In *ITiCSE-WGR '06: working group reports on innovation and technology in computer science education* (pp. 182–194). New York: ACM.

Meltzer, D.E. (2005). Relation between students' problem-solving performance and representational format. *American Journal of Physics*, *73*(5), 463–478.

Miller, G.A., & Chomsky, N. (1963). Finitary models of language users. In R.D. Luce, R.R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology*, (Vol. II pp. 419–491). New York: Wiley.

Mitrović, A., Suraweera, P., Martin, B., & Weerasinghe, A. (2004). DB-Suite: experiences with three intelligent, web-based database tutors. *Journal of Interactive Learning Research*, *15*(4), 409–433.

Monge, A.E., Fadjo, C.L., Quinn, B.A., & Barker, L.J. (2015). EngageCSEdu: engaging and retaining CS1 and CS2 students. *ACM Inroads*, *6*(1), 6–11.

Mullins, D., Deiglmayr, A., & Spada, H. (2013). Motivation and emotion in shaping knowledge co-construction. In M. Baker, S. Jarvela, & J. Andriessen (Eds.) *Affective learning together: social and emotional dimensions of collaborative learning* (pp. 139–161). Routledge.

Nouri, E., & Traum, D. (2014). Initiative taking in negotiation. In *15th annual meeting of the ACL Special interest group on discourse and dialogue* (pp. 186–193). Philadelphia.

Pirolli, P., & Anderson, J.R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology*, *39*(2), 240–272.

Pirolli, P., & Recker, M. (1994). Learning strategies and transfer in the domain of programming. *Cognition and Instruction*, *12*(3), 235–275.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming: what works? *Communications of the ACM*, *56*(8), 34–36.

Reese, D.S., Lee, S., Jankun-Kelly, T., & Henderson, L. (2014). Broadening participation in computing through curricular changes. In *ASEE southeast section conference*.

Renumol, V., Janakiram, D., & Jayaprakash, S. (2010). Identification of cognitive processes of effective and ineffective students during computer programming. *Transactions on Computing Education*, *10*(3), 1–21.

Resnick, L. (1989). *Knowing, learning, and instruction: essays in honor of rober glaser*. Hillsdale, NJ: Erlbaum.

Scime, A. (2008). Globalized computing education: Europe and the United States. *Computer Science Education*, *18*(1), 43–64.

Scott, T. (2003). Bloom's taxonomy applied to testing in computer science. In *Proceedings of the 12 annual CCSC rocky mountain conference*.

Searle, J.R. (1965). What is a speech act. In M. Black (Ed.), *Philosophy in America* (pp. 615–628). Ithaca: Cornell University Press. Reprinted in *Pragmatics*. A *Reader*, Steven Davis editor, Oxford University Press, 1991.

Searle, J.R. (1975). Indirect speech acts. In P. Cole, & J. Morgan (Eds.), *Syntax and semantics 3. Speech acts*. New York: Academic. Reprinted in *Pragmatics*. A *Reader*, Steven Davis editor, Oxford University Press, 1991.

Shah, F., Evens, M.W., & Michael, J.A. (2002). Classifying student initiatives and tutor responses in human keyboard-to-keyboard tutoring sessions. *Discourse Processes*, *33*(1), 23–52.

Soh, L.-K. (2006). Incorporating an intelligent tutoring system into CS1. In *SIGCSE'06* (pp. 486–490). Houston: Association for Computing Machinery.

Soller, A. (2004). Computational modeling and analysis of knowledge sharing in collaborative distance learning. *User Modeling and User-Adapted Interaction*, *14*(4), 351–381.

Soloway, E., & Spohrer, J.C. (1988). *Studying the novice programmer*. Mahwah, NJ: Lawrence Erlbaum Associates.

Walker, M., & Whittaker, S. (1990). Mixed initiative in dialogue: an investigation into discourse segmentation. In *Proceedings of the 28th annual meeting of the association for computational linguistics* (pp. 70–78). Morristown, NJ: Association for Computational Linguistics.

Wang, D., Han, H., Zhan, Z., Xu, J., Liu, Q., & Ren, G. (2015). A problem solving oriented intelligent tutoring system to improve students' acquisition of basic computer skills. *Computers & Education*, *81*, 102–112.

Wilson, B.C. (2002). A study of factors promoting success in Computer Science including gender differences. *Computer Science Education*, *12*(1–2), 141–164.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Winslow, L.E. (1996). Programming pedagogy—a psychological overview. *SIGCSE Bull.*, *28*(3), 17–22.

Xu, S., & Rajlich, V. (2004). Cognitive process during program debugging. In *Proceedings of the 3rd IEEE international conference on cognitive informatics, ICCI '04* (pp. 176–182). IEEE Computer Society.

Yang, F., & Heeman, P. (2010). Initiative conflicts in task-oriented dialogue. *Computer Speech & Language*, *24*(2), 175–189.

Yoon, B., & Garcia, O. (1998). Cognitive activities and support in debugging. In *Proceedings, 4th annual symposium on human interaction with complex systems* (pp. 160–169). IEEE.

Zweben, S., & Bizot, B. (2015). Relentless growth in undergraduate CS enrollment; doctoral degree production remains strong, but no new record (2013–2014 taulbee survey). *Computing Research News*, *27*(5).