Examining the Effect of Collaboration in an Online Learning Environment

Matthew Bojey
University of British Columbia
3333 University Way
Kelowna B.C.
1-250-212-7508
mbojey@gmail.com

ABSTRACT

At UBC Okanagan, an introductory Java programming course, COSC 121, is required for many Science students. In this course, several concepts are abstract and non-trivial for students to master. One of the concepts is a *linked list*, a data structure with interconnected nodes. In this project I designed and implemented a web based system to help students learn about linked lists in a collaborative setting. The system presents a visualization of linked lists and asks students to solve several exercises related to the operations on the data structure such as adding, deleting and reordering nodes. Students can choose to solve questions in the system either alone or in pairs using a synchronized online workspace and private chat room. The study focuses on examining the effect of student collaboration in Computer Science education piloted in the current COSC 121 class with approximately 120 students. In addition to finding significant performance and confidence increases in students, the project found that while each attempt takes longer collaboratively it is more efficient for students to solve these problems in pairs than alone. The system discussed in this paper can be found at http://young-refuge-6221.herokuapp.com/

Categories and Subject Descriptors

K.3.1 [**Computers And Education**]: Computer Uses in Education – *Collaborative Learning.* K.3.2 [**Computers And Education**]: Computer and Information Science Education – *Computer Science Education.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Computer Science Education, Linked Lists, Abstract Data Types, Collaborative Learning, System Evaluation.

1. INTRODUCTION

In general Science education many students are required to take a course in introductory programming. These courses cover many concepts including basics of variable assignment, control and conditional statements, object oriented programming and data structures. There has been extensive amounts of work done on ways to aid students in learning these and other related concepts [1]. Even with this existing work, for many students these more abstract concepts such as object oriented programming and data structures are challenging to understand and non-trivial to master.

I aim to help students understand one such data structure concept, *linked lists*. Linked lists are a data structure composed of interconnected nodes where each node contains data and one or more pointers to another node in the list. This project focuses on *singly linked lists* where each node only contains a pointer to the next node in the list. The list also has a *head* that points to the front of the list so that as programmers work with the list they have a place to start traversing the list.

In this project I designed and implemented a web based system for students to practice various operations on linked lists. The system was written using Ruby on Rails for the back end with JavaScript, HTML and CSS as the front end. The system allows students to work with graphical representations of lists and focus more on the conceptual ideas being operations rather than dealing with the details involved in programming them. The system also behaves how one would expect when programming in Java with an automated "garbage collector" that removes nodes from the list when there are no references to the node from elsewhere in the list. As the system is designed to be used by students who are still learning the material, the students may attempt a question as many times as they need in order to get a correct answer.

To examine the effect of collaboration as students use the system, I have two conditions during evaluation of the system, individual work and pair work. When students are working in pairs the actions of one student are reflected on the screen of the other student so that they are aware of what their partner is doing. There is also a built in private chat room so that partners can communicate via text chat. The system discussed in this paper can be found at http://young-refuge-6221.herokuapp.com/

2. RELATED WORK

As mentioned previously, there has been a large body of work done in Computer Science Education [1], of particular interest to this project is work done with linked lists and work done with collaboration. I will examine work done in these two areas next. To the best of my knowledge, there are no existing works on collaborative linked list education.

2.1 Linked Lists

Due to the challenging abstract nature of the linked list concept, existing work has focused on providing a graphical interface for students to interact with. One such program is *iList,* a program designed to instruct students on how to perform operations on linked lists [2]. The program combines graphical representations of lists and syntactically correct code to help students master the concept as well as become successful at implementing it. The focus of the program is to try to closely replicate the behaviors and outcomes of a human tutor by offering different levels of feedback to the user. The system has been tested and has been found to be effective and perceived to be useful and interesting by students. While the system uses graphical representations similar to the system I have designed they have no collaborative aspect to the system as all work is done individually. They did have a connection between the diagrams and the final code, which I did not have.

Another program that focuses on the graphical representation of lists to help explain the concept is *Java Visual Automated Linked List (JVALL)* [3]. This program works directly with the Java LinkedList class to show animations of what changes are happening to the data structure when different methods are run. The program supports many of the methods from the LinkedList class so that students can understand what is happening when they perform any operation on a linked list in their programs. This program uses the graphical representation to help reinforce the concepts of linked lists but it does not provide any exercises for students to work through like my system does.

A third program that uses the graphical representation of allow students to work with linked lists is presented by Panoiu et al. [4]. In this system, students work to perform basic operations on linked lists such as adding and removing nodes. This program is written as a stand-alone desktop application and does not support any collaborative aspects like the system implemented in this project does. This system, like *iList* does not support any collaborative work between students like my system does.

2.2 Collaborative Education

A number of studies have investigated the impact or collaboration in Computer Science education [5, 6. 7, 8]. Many of these studies involve splitting students into small groups and presenting them with problems similar to those that they would find in class or laboratory activities. In general, these activities do not allow students to directly work with each other and more encourage taking turns working on the problem and providing feedback to group mates. This is very different from my system where students work directly and synchronously with each other solving the same problem in tandem.

Collaborative education extends beyond the domain of just Computer Science and beyond the digital realm as well. In general, collaborative learning involves students working together to understand material or complete a task [9, 10]. There has been a large body of evidence that suggests that collaborative learning has a wide range of benefits including improved student performance, more positive attitudes towards the material and increased student retention [9, 10]. This has lead to the development of many computer systems to facilitate collaborative learning. One such system is *Lotfi Virtual Collaborative Learning (Lotfi VCL)* [11]. This system is a web-based system that was designed to facilitate collaboration between students in a variety of domains. While the system does not offer any domain specific educational frameworks the collaboration it offers has significant positive impacts on student engagement, attitude and performance. In contrast with my system, which is designed specifically for students learning about linked list, the *Lotfi VCL* is much more a collaboration framework where instructors who wish to use the framework develop domain specific information.

There has also been a large amount of work done on collaboration in Computer Science education in the past. In a 1997 study by Leidner and Fuller, it was shown that collaboration helped students feel that they understood the material better and increased interest in the material, specifically material in a Management Information Systems course [12]. This was done by allowing the students to form small groups to discuss the material presented in class before requiring them to perform individual assessments. However, this perceived understanding and increased interest did not translate into increased performance on the individual assessments. This study differs from my work as I created specific activities that students can work through as opposed to open discussion of small groups. Also, I limit the group size to only pairs of students and not larger groups.

One recent study by Beck and Chizhik has focused on collaboration in introductory Computer Science education [9]. In this study students would work in small groups while the instructor observes and at the end of the work sessions debriefs with the group to discuss their success and potential misunderstandings. The study showed significant increases in many areas of student learning when using a collaborative approach to Computer Science education as opposed to the typical individual approach.

2.2.1 Pair Programming

A technique common in Computer Science that is related to collaboration is pair programming. In pair programming, two programmers work in tandem with one member of the team typing the program or writing down a design while the other member is asking questions, spotting errors or thinking of alternatives. The roles are switched back and forth while the team works. This form of collaboration has been found to be very effective in both student and professional programmers [9]. While I do not enforce students to follow this paradigm I do want to acknowledge the existence and historical success of the style of pair collaboration.

3. LINKED LIST DOMAIN

As discussed above, a linked list is an abstract data type, a way to represent data that has specific behaviors that can be expected by an end user. The main idea behind a linked list is that items in the list are "linked" and can be access sequentially. Each item in a list is called a *node*. Each node contains some *data*, which can be anything including integers, sentences, or characters, and a *pointer*. The pointer is what links one node to the next in a list. The pointer stores the address of the next node in the list and by following the pointer one can reach the next node in the list. The end of the list is represented by a null pointer, or a pointer that points to nothing. A linked list starts with a *list head*. This is the only reference to the rest of the list that a student will have as they work with the list and serves as the starting point as the work through a list. The list head will point to the first node in the list.

The typical graphical representation of a linked list is boxes and arrows and that is what I will use here as well. A box that has two sections represents a node. The blue section will represent the data that is stored in the node and the red section will represent the pointer. The pointers will either have an arrow pointing to the next node or a null pointer represented by a ground symbol. I also use a purple box to represent the list head. This structure can be seen in Figure 1.

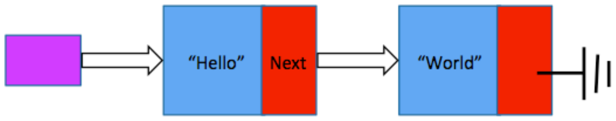


Figure 1: A graphical representation of a linked list with 2 nodes.

To effectively master linked lists students must be able to understand the static representation of the data presented above as well as be able to complete several operations on the data structure. These operations can include adding a node, removing a node, finding a node with a specific value, traversing all the nodes in a list, or reordering the nodes in a list. These operations involve a deep understanding of the underlying structure as well as careful planning of the operations. In the process of performing these operations students must mentally keep track of where all the pointers in the list are pointing to and make sure that they aren't skipping over any nodes. In the Java programming language this is especially important, as the garbage collector will simply delete any nodes that don't have anything pointing to them during the process. This is why a graphical representation has often been used and is so helpful to understanding the concepts behind these operations. For example, see Figure 2 for the process of adding a node to a list.

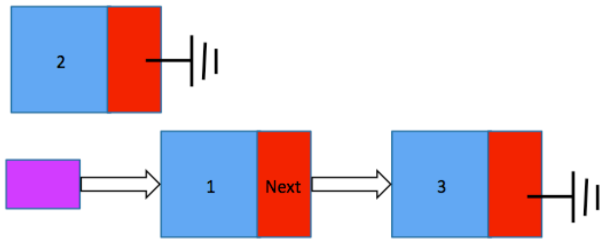


Figure 2 a): A node that will be inserted between the 2 nodes in the existing list.

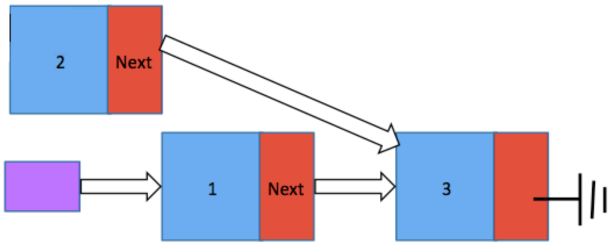


Figure 2 b): First I set the next pointer to the node that will come after the node to be inserted.

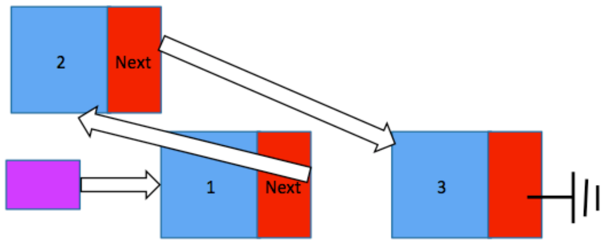


Figure 2 c): Next I update the next pointer of the node that will come before the node to be inserted.

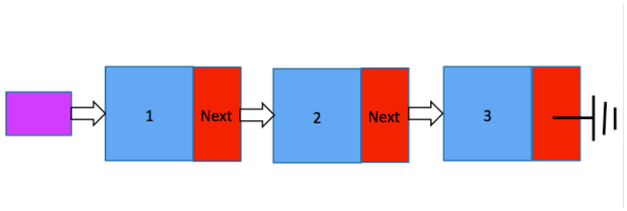


Figure 2 d): Finally I have the list after the node has been inserted.

4. SYSTEM DESIGN

As mentioned previously, the system was designed to be a web application. The framework that was used was Ruby on Rails as it facilitates a relatively easy and straightforward development through the use of various publically available add-ons called Gems. The system was designed with several features as well as focusing on several operations involving linked lists.

4.1 System Features

In order to facilitate the individual and collaborative modes of the system several features were required. Also, to help with evaluation of the system several logging and reporting features were developed as well.

4.1.1 User Accounts

In order to track students' performance and be able to give them credit for their work user accounts needed to be created. This was done with the use of the Devise gem. The user accounts were also monitored so that other users would know when a user was online and available to collaborate with. This was done with the use of Asynchronous JavaScript and XML (AJAX) to make a call from the users browser to the server every 5 seconds to update that they were online. Then when another user was checking who was online, the system would look for all users who were update as online less than 5 seconds ago to populate the list. All users were required to create an account to use the system and prior to creating an account were presented with terms of their use of the system as per Research Ethics Board (REB) standards.

4.1.2 Tutorial

The first feature that a user would see when they begin using the system is a tutorial. This tutorial was designed to familiarize the user with how lists will be represented in the system. It gives a rudimentary overview of what a list is and how is it represented but it is not meant to replace classroom instruction. It is assumed that students have been exposed to the concept of linked lists prior to starting to use the system. The tutorial also gives students a chance to play with the nodes and pointers in a list before beginning the practice exercises.

4.1.3 Practice Questions

The main piece of the system is the practice questions that are presented to the students. These questions can all be attempted either individually or in pairs. The questions presented are similar in scope to those in Figure 2 but cover different topics as shown in Section 4.2. The question workspace is programmed in JavaScript using the Kinetic.js open source library. The library allows to the creation and manipulation of clickable and draggable shapes within a certain workspace called a Scene. Each question that a student works on creates a new Scene in the server database and each Scene has associated Nodes and Actions. Each time that a student performs an action, such as moving a node to another location or adding a temporary variable to the scene, an AJAX call is made to the server to update the current Scene in the database. When a student feels that they have completed the required steps in the questions they can submit the question and the Scene is then checked for correctness. The Scene is checked by first determining what question the student was working on then checking to make sure that the criteria laid out by the question is correct. For example, if the system were checking the question presented in Figure 2 it would start by making sure that there were 3 nodes in the list. Then it would make sure that the nodes are in order. If either of these checks would fail, a notification would be given to the student letting them know that they have made a mistake and what that mistake is.

When a student is working on a question and they submit the correct answer they receive 4 points if done individually and 5 points if done with a partner. Each practice question also has an optional hint available and if the hint is used the student receives only 2 points upon completing the question if they are working individually or collaboratively. The student will receive 100% for the activity after gaining 80 points. Students were allowed to attempt a question as many times as they want with no penalty to their score for incorrect answers. This is due to the fact that the system was designed as a practice tool and not an evaluation tool. Students were allowed to attempt questions in any order that they choose. This scoring system was developed in collaboration with the current instructor of COSC 121, the course that the system was tested in. A full list of the questions presented in the system can be seen in the Appendix.

The workspace as seen by the student while completing a question can be seen in Figure 3.

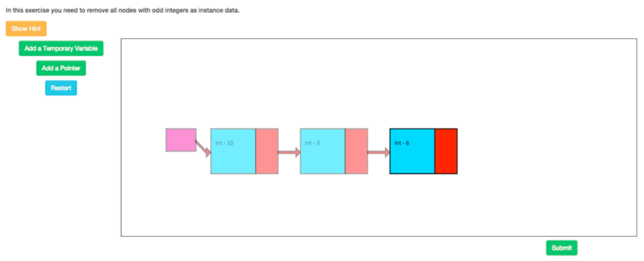
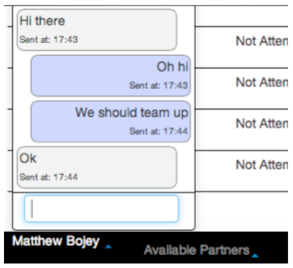

In this exercise you need to remove all nodes with odd integers as instance data.

Figure 3: The workspace as seen by a student when solving an exercise.

4.1.4 Chat and Collaborative Exercises

When a student wants to work with a partner, they first find an available partner from the online users. They then can work on questions with their partner. While working on questions they can also use a built in text chat system to discuss issues or ideas related to the problem. This text chat is built so that when a student sends a message, that message is sent to the server and a new message is created in the database. Then every 5 seconds the user's browser will check to see if there are any new messages sent from their partner but making an AJAX call to the database and retrieving any new messages. These messages will then be displayed to the user. In this way, the chat is not instant but with only a 5 second delay user testing showed that this was not an issue. The chat system can be seen in Figure 4.


Hi there
Sent at: 17:43

Oh hi
Sent at: 17:43

We should team up
Sent at: 17:44

Ok
Sent at: 17:44

Figure 4: The chat system interface.

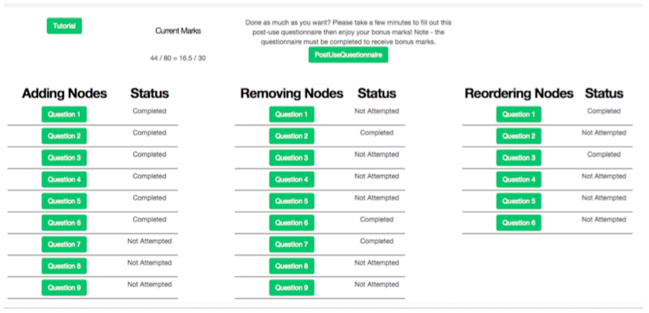
When a student begins working on a question and they have chosen a partner, an invitation to join will be sent to the partner. Then partner can then accept the invitation and will be able to see what the current user is working on. When one partner makes a change to the Scene the other partner will see the change 1 second later. This is done by having each partner constantly checking with the database to make sure that they current version of the Scene displayed on their screen is the most up to date possible. As mentioned earlier, due to the fact that every action is recorded to the database as it happens by each user then this look up becomes very simple to implement with an AJAX call to get the most current Scene and a JavaScript function to move the nodes around on the screen in the Kinetic.js workspace. Currently, there is no way for a user to know that their partner has selected a node and plans to move it but this is an improvement that could be made in the future.

4.1.5 Pre and Post Use Questionnaires

To evaluate student confidence and attitudes towards the material pre and post use questionnaires were also developed. The pre use questionnaire was presented to student before they began using the system and students completed the post use questionnaire when they were finished with the system. They post use questionnaire could be completed at any time if the student wanted to stop using the system before receiving full marks. In addition to questions from the pre use questionnaire pertaining to confidence and attitude the post use questionnaire also presented questions about the usability of the system to the students.

4.2 Educational Design

The system was designed as a way for student to practice with the concepts that they had been presented in class, it was not designed to be the first and only experience students have with linked lists. As such, the system was designed with 3 mains types of questions, adding nodes, removing nodes and reordering lists. Students were able to select any question to do at any time. The interface presented to students to select a question is shown in Figure 5.



44 / 80 = 16.5 / 30

Done as much as you want? Please take a few minutes to fill out this post-use questionnaire then enjoy your bonus marks! Note - the questionnaire must be completed to receive bonus marks.

Adding Nodes	Status	Removing Nodes	Status	Reordering Nodes	Status
Question 1	Completed	Question 1	Not Attempted	Question 1	Completed
Question 2	Completed	Question 2	Completed	Question 2	Not Attempted
Question 3	Completed	Question 3	Not Attempted	Question 3	Completed
Question 4	Completed	Question 4	Not Attempted	Question 4	Not Attempted
Question 5	Completed	Question 5	Not Attempted	Question 5	Not Attempted
Question 6	Completed	Question 6	Completed	Question 6	Not Attempted
Question 7	Not Attempted	Question 7	Completed		
Question 8	Not Attempted	Question 8	Not Attempted		
Question 9	Not Attempted	Question 9	Not Attempted		

Figure 5: The interface presented to students to select what question to work on.

4.2.1 Adding Nodes

Students were presented with a choice of 9 questions related to adding nodes. There were 3 sub-types of problem with 3 questions each: adding a node to the front of a list, adding a node to the middle of a list, and adding a node to the end of a list. Each question would contain a slight variation in things such as the initial number of nodes in the list or the type of data stored in the list. These variations were to help students see that the process of adding a node to a list is independent of such factors and is the same every time.

4.2.2 Removing Nodes

Students were also presented with a choice of 9 questions related to removing nodes. There were 3 sub-types of problem with 3 questions each: removing a node from the front of a list, removing a node from the middle of a list, and removing a node from the end of a list. Again, each question would contain a slight variation in things such as the initial number of nodes in the list or the type of data stored in the list. Removing nodes from a list is typically a harder question as it is easy to remove the entire list and this difficulty is represented in the empirical results discussed in Section 6.

4.2.3 Reordering Nodes

Students were presented with a choice of 6 questions related to reordering nodes. There were 2 sub-types of problem with 3 questions each: reversing the order of a list and sorting a list according to some ordering. Again, each question would contain a slight variation in things such as the initial number of nodes in the list or the type of data stored in the list. These were the most difficult questions in the system, as the students would need to keep track of several pointers that needed to be moved to order the list properly. These exercises were designed to help students realize that temporary variables may be needed to hold on to parts of the list while they would work with the rest of the list. Again, the empirical results show that these questions were indeed the most difficult.

5. EVALUATION METHODOLOGY

Prior to testing with students, two primary rounds of testing were completed. First user interface (UI) testing was done. This was done to ensure that the interface was easy to use and made sense. This was done with 2 students who had taken the course the previous year so the material was not a challenge and they were able to focus on the usability of the system. As a result of these tests changes were made to the tutorial to more clearly explain the representation of linked lists used in the system. These changes were a more step-by-step introduction to the components of a linked list and how they are represented in the system. An interactive sandbox mode was also introduced in the tutorial so that students can see how the addition or movement of nodes and arrows works with the system. These changes made the students able to focus on the material when they start working on the problems and not need to learn the representation and interactions present in the system. Also, the chat window was moved from the bottom right corner to the bottom left corner as it was found to obscure the workspace on smaller screens.

Secondly, a small pilot test was completed with several upper level computer science students. Again, experienced students were asked to participate, as they would be able to find flaws in the system without having to worry about solving the problems that were easy for them. These test involved 6 participants, 4 of whom worked in pairs and 2 who worked alone. The participants were able to find a few exceptional cases where the evaluation of questions was incorrect as well as make suggestions to improve the responsiveness of the chat and collaboration system. As a result of these tests the verification issues were solved and the students in the COSC 121 course could use a cleaner more responsive system when they tested it. Removing the time between updates when a partner would move a node in the linked list changed the responsiveness. The chat was also made more responsive by decreasing the time between calls to the database to check for new messages from a partner.

As mentioned previously, the system was evaluated with the current, Winter 2014 Term 2, class of COSC 121 at UBC Okanagan. There were 120 students enrolled in the class and 67 chose to participant in the student. Of the 67 participants 16 worked in pairs and 51 worked individually. This was due to the fact that the system was presented in class as an optional bonus assignment. The system was presented to students online and they were allowed to work on the system any time during the week. Due to this open ended nature of the system use, which was necessary to increase participation; we were not able to structure the amount of students who would work collaboratively or individually. There were no other options for participation as lecture time was unavailable and lab sessions had very low attendance. A student would sign up for an account and complete the pre use questionnaire. The student would then be presented with the tutorial, which all students were required to do individually. Then the student would be allowed to select what questions to work on and whether they wanted to work alone or with a partner. They would be told how many points they currently had and when they were done they would complete the post use questionnaire. If a student worked individually and did not use any hints they would only need to complete 20 of the 24 questions and this was how most students used the system.

On average, students spent 3.45 minutes to complete the pre-use questionnaire, 6.23 minutes to complete the tutorial, 7.78 minutes to complete the post-use questionnaire and 26.34 minutes solving the exercises in the system. Of the students who completed the exercise, all but one student completed enough questions to receive the full 80 marks. The one student who did not receive the full 80 marks received 66 marks on 17 questions done individually.

6. RESULTS

After testing the system with students, I are able to see that overall students' confidence increased, they found the system enjoyable and interesting, students' performance increased and that solving questions as a pair was more effective than solving them alone. I will investigate each of these statements more in detail below.

6.1 Student Confidence

As discussed in Section 4.1.5, pre and post use questionnaires were developed and presented to the students. These questionnaires were designed to determine if student confidence about linked list operations improved as a result of using the system. All values reported from the questionnaires correspond to Likert Scale average with 1 - Strong Disagree 5 - Strongly Agree. Overall I see that there is a significant ($p < 0.01$ using a two-tailed T test) increase in student confidence in the material after using the system. This can be seen in Figure 6.

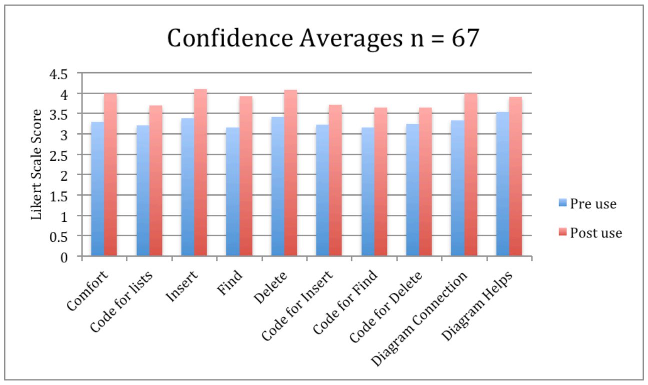


Figure 6: Average confidence scores for pre and post use questionnaires. Average pre use confidence = 3.30, average post use confidence = 3.87. Questions asked can be found in Section 6.1.1.

I also see that there are significant ($p < 0.01$ using a two-tailed T test) increases in confidence in both collaborative and individual students as well in Figures 7 and 8.

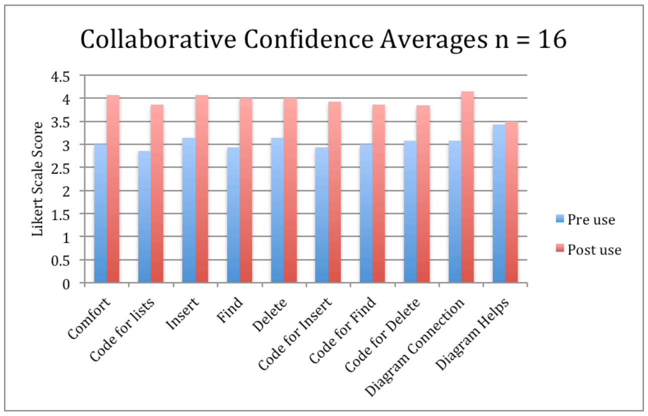


Figure 7: Average confidence scores for pre and post use questionnaires for students who worked collaboratively. Average pre use confidence = 3.05, average post use confidence = 3.92.

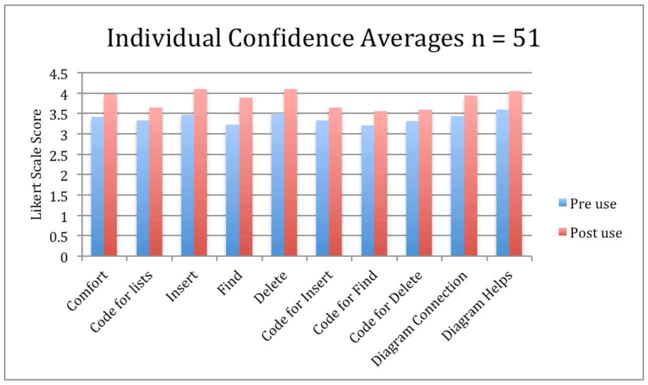


Figure 8: Average confidence scores for pre and post use questionnaires for students who worked individually. Average pre use confidence = 3.05, average post use confidence = 3.92.

I also see, in Figure 9 that the average confidence increase for students who worked collaboratively is significantly higher than those who worked indiviually ($p < 0.01$ using a two-tailed T test).

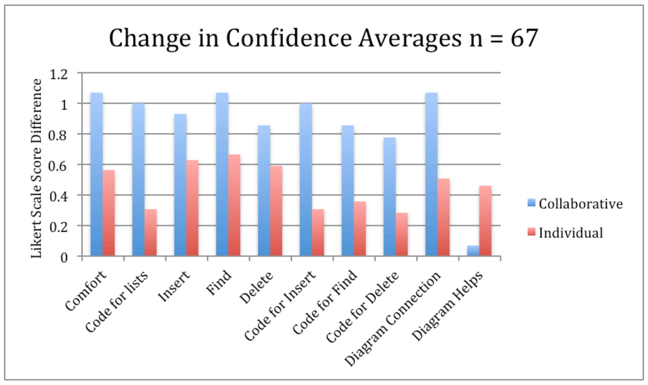


Figure 9: Average increase in confidence after using the system via difference in Likert Scale averages. Average confidence gain for collaborative student = 0.87, average confidence gain for individual student = 0.47.

6.1.1 Questions Asked in Confidence Questionnaire

The questions asked to students in both the pre and post use questionnaires to assess their confidence in the material were as follows.

1. I am comfortable with using linked lists:
2. I am confident in my ability to write code to make linked lists:
3. I am confident in my ability to insert into linked lists:
4. I am confident in my ability to find a given node in a linked list:
5. I am confident in my ability to delete from linked lists:
6. I am confident in my ability to write code to insert into linked lists:
7. I am confident in my ability to write code to find a given node in a linked lists:
8. I am confident in my ability to write code to delete from linked lists:
9. It is easy for me to see the connection between the diagrams for lists and the code for them.
10. Drawing list operations helps me understand them better.

6.2 System Usability

In the post use questionnaire students were also asked about the usability of the system. Overall, students found the system easy to use and useful. However, students who worked individually found the system significantly more usable than those who worked collaboratively ($p < 0.01$ using a two tailed T test). These results can be seen in Figure 10.

Overall these results show that more work needs to be put into making the collaborative mode as user friendly as possible. More work may be required to continue increasing the response time and communicating between partners what each other are doing. This could be done with a shorter refresh time as well as implemented a feature to highlight a node when a student's partner has selected it to cut down on situations where both students are fighting over where to move a node.

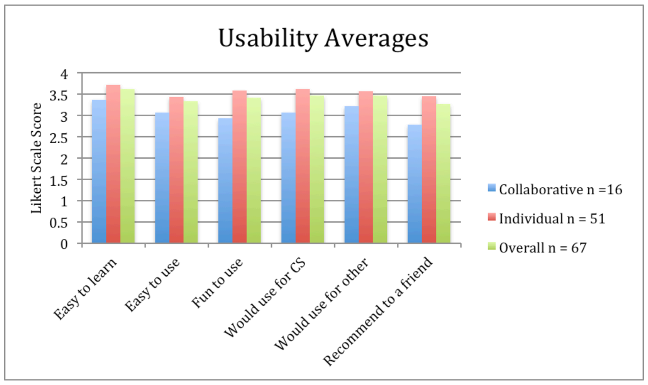


Figure 10: Average usability scores after using the system. Average score overall = 3.45, average score for collaborative student who collaborated = 3.07, average score for individual student = 3.56. Questions asked can be found in Section 6.2.1.

6.2.1 Questions Asked in Usability Questionnaire

1. When I first used the system, it was easy to learn it and figure out how to use it.
2. I found the system very easy to use.
3. I found the system fun to use.
4. I would use the system to help study for Computer Science.
5. If the system had exercises for other subjects (e.g., Math), I would use it for them too.
6. I would recommend it to my friends to use the system.

6.3 Student Performance

In evaluating the system I looked at performance along 2 separate metrics. First I looked at the number of attempts a student would take to answer a question correctly. This is available to us due to the design of a practice system and not an evaluation system. This allows the student to try a question as many times as they need to get a correct answer. The second metric I used was how long a student would spend to get submit an answer. I then looked at these metrics with the 2 conditions, collaborative pair work and individual work.

6.3.1 Student Attempts per Question

I see in Figure 11 the average number of attempts per question for students working individually. I can see that questions 1-9, the adding node questions, on average take the least number of attempts, removing node questions, 10-18, take the second most and the reordering questions, 19-24, take the most attempts. This is what I had expected when I designed the system and makes sense given that this is the order of increasing difficulty. I can see a similar pattern in Figure 12 for students who worked in pairs but I have no data for questions 23 and 24 as no students completed these in pairs. This is due to the fact that more marks were offered per question when working in pairs and so these questions were never required.

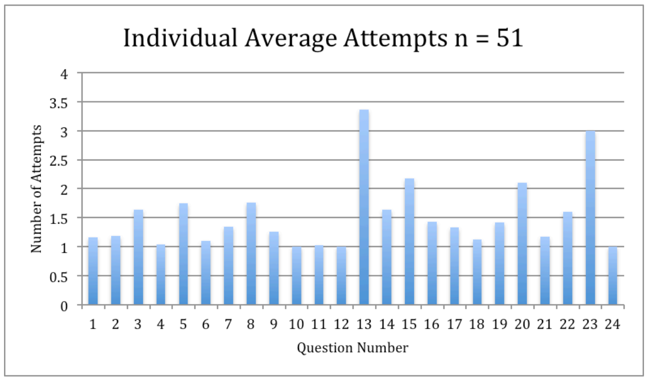


Figure 11: Average number of attempts per question until a correct answer is submitted for individual students. Average for adding nodes questions = 1.35, average for removing nodes questions = 1.56, average for reordering nodes questions = 1.72

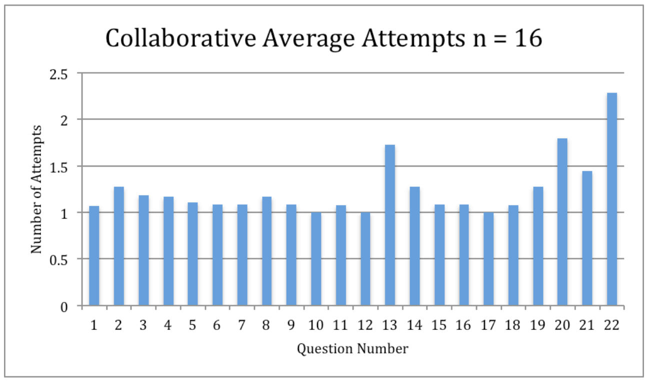


Figure 12: Average number of attempts per question until a correct answer is submitted for collaborative students. Average for adding nodes questions = 1.13, average for removing nodes questions = 1.15, average for reordering nodes questions = 1.70

The other result that I see when looking at average number of attempts per question is shown in Figure 13 and Figure 14. I see that it take significantly fewer attempts to get a question correct collaboratively than individually ($p < 0.05$ with a two-tailed T test).

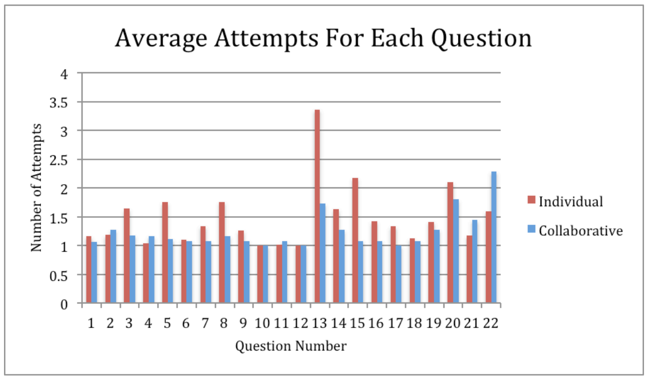


Figure 13: Average number of attempts per question until a correct answer is submitted for individual and collaborative students.

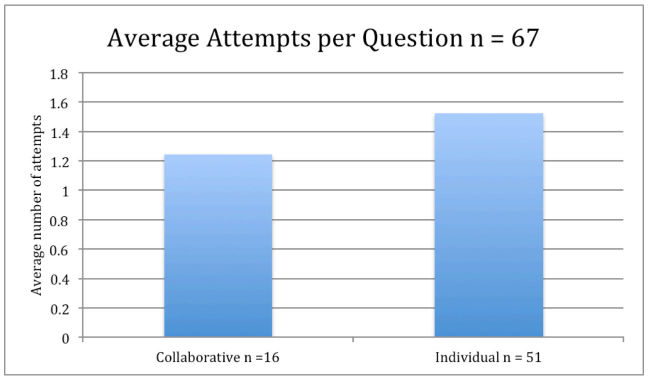


Figure 14: Average number of attempts until a correct answer is submitted for individual and collaborative students. Average for individual students = 1.52, average for collaborative students = 1.24

6.3.2 Time per Question

The final metric that I used to evaluate the system was the amount of time student spent on questions. I decided to split this into 2 sub categories, the amount of time to submit a correct answer and the amount of time to submit an incorrect answer. I see if Figure 15 and Figure 16 that the time it takes a student to submit a correct answer decreases significantly ($p < 0.01$ using a two-tailed T test).

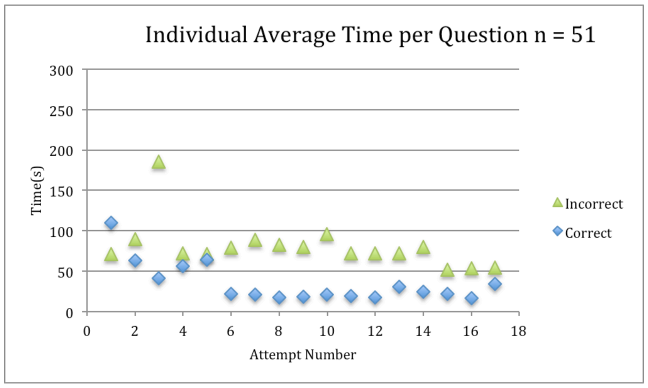


Figure 15: Average time per question for individual students. Average time per incorrect question = 79.38s, average time per correct question = 35.70s.

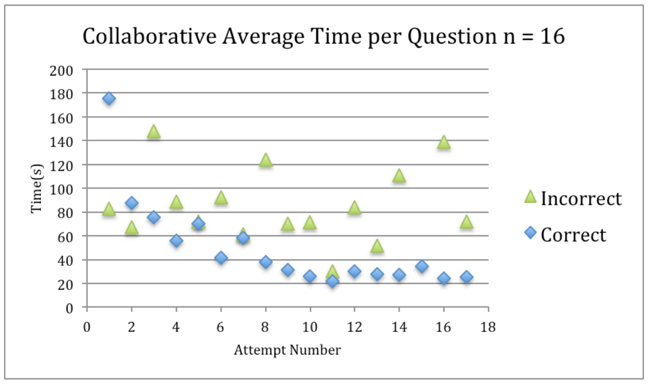


Figure 16: Average time per question for collaborative students. Average time per incorrect question = 83.39s, average time per correct question = 53.51s.

By looking at the data present in Figure 15 and Figure 16 I also see that the amount of time it takes to submit a correct answer when collaborating is significantly longer than the time to submit a correct answer when working alone ($p < 0.01$ using a two-tailed T test). I do not see any significant difference in the time to submit an incorrect answer however. By using this information as well as the average number of attempts from Figure 14 I can see that while it takes longer per attempt when working collaboratively, because it requires fewer attempts it is actually more efficient to work with a partner. This data is presented in Table 1.

Table 1: The average time it takes to complete a question correctly individually and collaboratively. This was calcuated by taking the average time per correct attempt (T_C) and adding average number of attempts(A) -1 * average time per incorrect attempt (T_I). $T_C + (A-1)*T_I$

	Avg. # of Attempts	Time per Correct Attempt	Time per Incorrect Attempt	Time for Correct Answer
Collaborative	1.24	53.51s	83.39s	72.92s
Individual	1.52	35.70s	79.38s	76.08s

7. CONCLUSIONS AND FUTURE WORK

Overall, I have shown that the system I created was able to significantly increase student confidence and performance. I also have shown that the confidence gained from using the system is significantly higher when working with a partner. I have also shown that while it takes significantly longer to complete an attempt with a partner, the fact that significantly less attempts are needed means that it is still more efficient to work with a partner when attempting conceptual linked list questions in the system. This is in agreement with existing work in the field of collaborative education and I feel that these results show there is a need for more systems like this in the field to help with other conceptual challenging topics in the sciences.

In the future I would like to improve the responsiveness of the collaboration system and add new ways for collaborators to communicate more effectively. Some ways that I could do this would be built in messages when starting to move a node or complete an action or integrating existing voice or even video chat software.

Also, although it became beyond the scope of the project I also think that adding a way to draw the graphical representation of linked lists back to implemented code would be beneficial as well. This would allow students to understand the concepts and then test what they have learned by trying to program them in Java.

8. ACKNOWLEDGEMENTS

I would like to thank Dr. Bowen Hui for her supervision and guidance throughout the project. I would like to thank Dr. Abdallah Mohamed for his support and suggestions as I tested the system with his COSC 121 class. I would also like to thank all those who helped with UI and pilot testing. Finally I would like to thank the students of COSC 121 for using the system, providing feedback, and their patience as some early bugs were ironed out.

9. REFERENCES

[1] Randolph, Justus, et al. "A methodological review of computer science education research." *Journal of Information Technology Education: Research*7.1 (2008): 135-162.
[2] Fossati, Davide, et al. "Supporting computer science curriculum: Exploring and learning linked lists with

iList." *Learning Technologies, IEEE Transactions on*2.2 (2009): 107-120.

[3] Dershem, Herbert L., Ryan L. McFall, and Ngozi Uti. "Animation of Java linked lists." *ACM SIGCSE Bulletin*. Vol. 34. No. 1. ACM, 2002.

[4] Panoiu, Manuela, et al. "An interactive learning environment for analyze linked list data structures." *International Journal of Computers, Communications & Control* 1 (2006): 355-359.

[5] Chase, Joe D., and Edward G. Okie. "Combining cooperative learning and peer instruction in introductory computer science." *ACM SIGCSE Bulletin*. Vol. 32. No. 1. ACM, 2000.

[6] Falkner, Katrina, and Edward Palmer. "Developing authentic problem solving skills in introductory computing classes." *ACM SIGCSE Bulletin*. Vol. 41. No. 1. ACM, 2009.

[7] Gonzalez, Graciela. "A systematic approach to active and cooperative learning in CS1 and its effects on CS2." *ACM SIGCSE Bulletin* 38.1 (2006): 133-137.

[8] Joseph, Anthony, and Mabel Payne. *Group dynamics and collaborative group performance*. Vol. 35. No. 1. ACM, 2003.

[9] Beck, Leland, and Alexander Chizhik. "Cooperative learning instructional methods for CS1: Design, implementation, and evaluation." *ACM Transactions on Computing Education (TOCE)* 13.3 (2013): 10.

[10] Johnson, David W., Roger T. Johnson, and Karl A. Smith. "Active learning: Cooperation in the college classroom." (1991).

[11] Lotfi, Zahra, et al. "Collaborative E-learning tool for secondary schools." *Journal of applied sciences* 13 (2013): 22-35.

[12] Leidner, Dorothy E., and Mark Fuller. "Improving student learning of conceptual information: GSS supported collaborative learning vs. individual constructive learning." *Decision Support Systems* 20.2 (1997): 149-163.

10. APPENDIX - LIST OF EXERCISES

Here are all the exercises presented to students in the system. Students were allowed to work through the exercises in any order they choose.

1. In this exercise you need to take the unconnected node and add it to the list as the first node. Make sure that when you are done the list has 3 nodes.
2. In this exercise you need to take the unconnected node and add it to the list as the first node. Make sure that when you are done the list has 5 nodes.
3. In this exercise you need to take the unconnected node and add it to the list as the first node. Make sure that when you are done the list has 3 nodes and that those nodes are still in increasing numerical order.
4. In this exercise you need to make a new node and add it to the list so that the nodes are in ascending numerical order. Make sure that when you are done the list has all 3 nodes.
5. In this exercise you need to take the unconnected node and add it to the list so that the nodes are in ascending numerical order. Make sure that when you are done the list has all 3 nodes.
6. In this exercise you need to take the unconnected node and add it to the list so that the nodes are in alphabetical order. Make sure that when you are done the list has all 3 nodes.
7. In this exercise you need to take the unconnected node and add it to the end of the list. Make sure that when you are done the list has all 3 nodes.
8. In this exercise you need to add the unconnected node to the end of the list. Make sure that when you are done the list has all 5 nodes.
9. In this exercise you need to take the unconnected node and add it to the list making sure that the list is still in numerical order. Make sure that when you are done the list has all 3 nodes.
10. In this exercise you need to remove the front node from the list.
11. In this exercise you need to remove the node in the list with the smallest integer as instance data.
12. In this exercise you need to remove the front node from the list.
13. In this exercise you need to remove all nodes with odd integers as instance data.
14. In this exercise you need to remove all nodes with even integers as instance data.
15. In this exercise you need to remove all nodes with no instance data.
16. In this exercise you need to remove the last node from the list
17. In this exercise you need to remove the last node from the list
18. In this exercise you need to remove the last node from the list
19. In this exercise you need to reverse the order of the nodes in the list. That is, make the nodes in the list be sorted by ascending numerical order.
20. In this exercise you need to reverse the order of the nodes in the list. That is, make the nodes in the list be sorted by descending numerical order.
21. In this exercise you need to reverse the order of the nodes in the list. That is, make the nodes in the list be sorted by alphabetically.
22. In this exercise you need to sort the given nodes in the list. The list needs to be in alphabetical order.
23. In this exercise you need to reorder the nodes in the list. The list needs to be in numerically ascending order.
24. In this exercise you need to rearrange the nodes in the list so that they are in alphabetical order.