Active Learning During Lecture Using Tablets

Barry L. Kurtz
James B. Fenwick
Rahman Tashakkori
Appalachian State University
Boone, NC 26808
828-262-2370
{blk,jbf,rt}@cs.appstate.edu

Ahmad Esmaili
Stony Brook University
Stony Brook, NY 11794-4400
631-632-1628
esmaili@cs.stonybrook.edu

Stephen R. Tate
University of North Carolina
at Greensboro
Greensboro, NC 27402
336-256-1112
srtate@uncg.edu

ABSTRACT

Closed labs have provided hands-on experience for students under supervised conditions. Microlabs extend this approach into the lecture format with very short hands-on activities in the "middle of the lecture." We have tested this approach using tablets as small as 7 inches; student laptops can also be used. Logical microlabs allow students to solve a conceptual problem in computer science that does not involve any programming. Solutions are entered graphically and submitted for evaluation. Code magnet microlabs allow students to construct a method to solve the same problem programmatically. The code is compiled and, if successful, evaluated by a sequence of unit tests. These are integrated into a Microlab Learning Cycle of exploration (the logical microlab), invention (the instructor guides the students toward an algorithm during lecture), and application (the code magnet microlab). This approach has been evaluated at three universities and proven to be popular with students and educationally effective. This work is supported, in part, by three National Science Foundation grants.

Categories and Subject Descriptors K.3.2 [Computer and Information Science Education] Computer Science Education

Keywords active learning, automated grading, tablets

1. APPROACHES TO ACTIVE LEARNING

A microlab is a 5-10 minute activity that occurs during lecture, students can work individually or in pairs, answers are submitted to an automated grading system, students are given constructive feedback if the submitted solution is not correct, modified solutions can be re-submitted for evaluation. The use of microlabs is based on the Learning Cycle model of instruction.

1.1 The Learning Cycle

The Learning Cycle was developed by Robert Karplus in the 1960s as part of the Science Curriculum Improvement Study [4]. This model of instruction has three phases:

1. Exploration – after a brief introduction to the topic students are asked to explore a particular problem and see whether they can discover a solution. This is guided discovery in the sense that as students work on the problem they receive feedback that guides them towards a correct solution.
2. Invention – once students have experienced trying to solve one or more problems in the target topic, the instructor, supported by the textbook or other learning resources, introduces the topic, as is commonly done in class lectures.
3. Application – Once the students are introduced to the topic both experientially (the exploration phase) and formally (the invention phase) they apply this knowledge to related problems in the domain. For computer science students this often means writing a program or methods in a program that allows the computer to solve problems in the target domain.

SIGCSE'14, March 5–8, 2014, Atlanta, Georgia, USA.
Copyright © 2014 ACM 978-1-4503-2605-6/14/03…$15.00.
http://dx.doi.org/10.1145/2538862.2538907

1.2 Automated Grading Systems

Ken Bowles at the University of California San Diego introduced the use of automatically-graded programming quizzes to support instruction in UCSD Pascal in the CS1 course in the late 1970s [2]. Kurtz has used this approach at three universities: the University of California Irvine, New Mexico State University [7], and Appalachian State University [6].

WEB-CAT is a grading system developed at Virginia Tech that evaluates student programs on three criteria: a test validity score, a test correctness score, and a code correctness score. [3]

Praktomat [12] is a web-based manager for programming labs; Praktomat automatically compiles, tests and checks the submitted programs; submissions not meeting the quality criteria are rejected.

Our automated grading of code magnet microlabs takes a similar approach to those noted above. We extend this approach to conceptual problems by automatically grading logical microlabs where students have entered a solution to a problem graphically, such as constructing a binary tree with nodes and edges.

1.3 Visual Programming Languages

A visual programming language (VPL) allows users to create programs by manipulating program elements graphically rather than by specifying them textually. Three widely used examples are:

- Alice, a freeware object-oriented educational programming language with an integrated development environment (IDE) that uses a drag and drop environment to create computer animations using 3D models. [1]
- Scratch, created by the MIT Media Lab, is intended to motivate learning in pre-college students through experimenting and creating projects, such as interactive animations, games, etc. [9]
- AppInventor, also from MIT, is a VPL for building mobile applications targeting the Android platform [8].

Our code magnets interface is similar to a visual programming language in using a drag and drop approach to compose program code for methods but does not attempt to provide the open-ended flexibility of a VPL.

2. THE MICROLAB LEARNING CYCLE

The Microlab Learning Cycle includes exploration (the logical microlab), invention (the instructor guides the students toward an algorithm), and application (the code magnet microlab). In the subsections that follow, we will describe each phase in the context of a problem involving binary trees.

2.1 Exploration – the logical microlab

A logical microlab asks a student to solve a conceptual problem in computer science. The problem does not involve any programming. Solutions are entered graphically and submitted for evaluation. To illustrate this process consider the following microlab where a student is asked to build a binary tree given the preorder and the inorder traversals (see Figure 1).

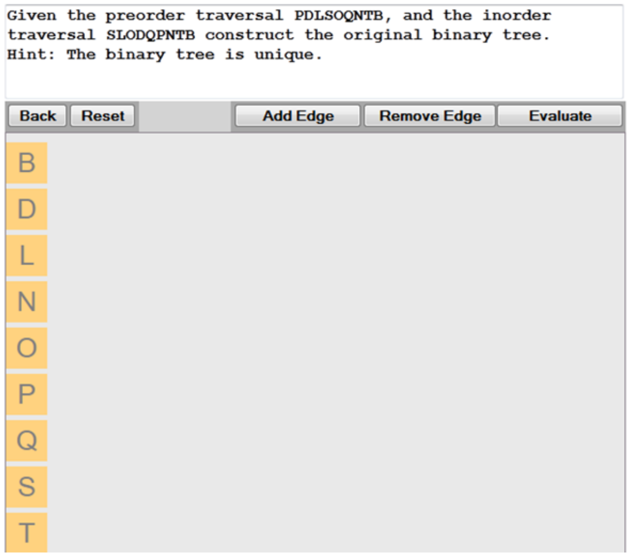
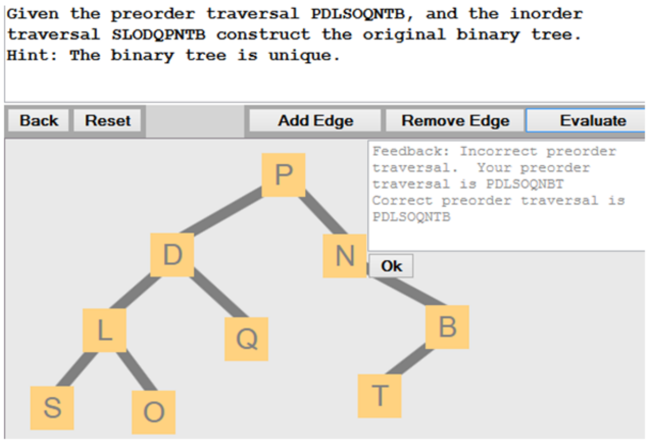

Given the preorder traversal PDLSOQNTB, and the inorder traversal SLODQPNTB construct the original binary tree.
Hint: The binary tree is unique.

Figure 1: Initial Screen

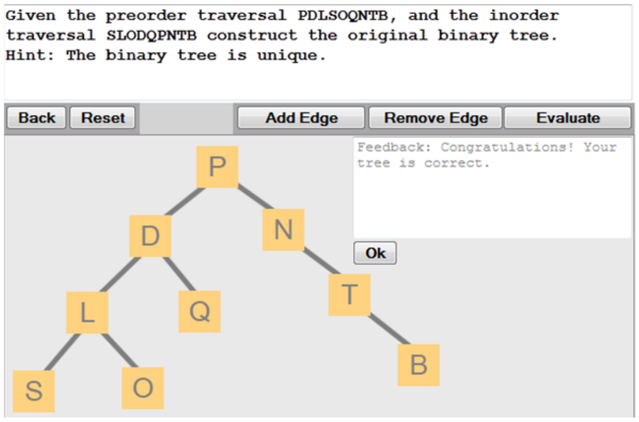
A solution that is close to correct is shown in Figure 2.


Given the preorder traversal PDLSOQNTB, and the inorder traversal SLODQPNTB construct the original binary tree.
Hint: The binary tree is unique.

Feedback: Incorrect preorder traversal. Your preorder traversal is PDLSOQNBT Correct preorder traversal is PDLSOQNTB

Figure 2: An Incorrect Solution

Notice that the feedback is designed to guide a student to the correct solution. The student can quickly spot that the nodes are mostly correct except for B and T. Adjustment of these nodes leads to the correct solution shown in Figure 3. If students do not find a correct solution in the allotted time, the instructor asks them to stop working on the problem and return their attention to lecture. Some students will not complete the activity in the time allowed, so all attempts are saved by the system allowing students to complete the problem outside of class. We have found that if students are given full credit for solutions submitted by the start of the next lecture period, they are willing to put down unfinished work and engage in lecture again.


Given the preorder traversal PDLSOQNTB, and the inorder traversal SLODQPNTB construct the original binary tree.
Hint: The binary tree is unique.

Feedback: Congratulations! Your tree is correct.

Figure 3: A Correct Solution

2.2 Invention – developing the algorithm

Building on the experiences with the logical microlab, it is the instructor's job to guide students toward an algorithm that will build the binary tree based on the preorder and inorder traversals. An instructor can begin by eliciting observations from the logical microlab activity, such as those below.

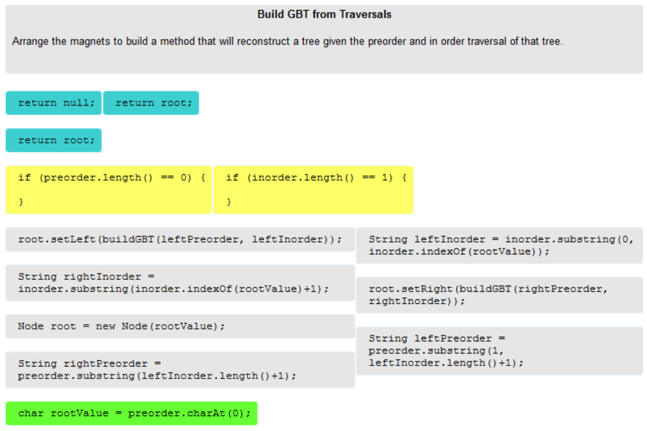
- The first node in the preorder traversal is the root of the tree.
- This node divides the inorder traversal into the inorder traversal of the left subtree and the inorder traversal of the right subtree.
- Using the lengths of these two inorder traversals, the left and right preorder traversals can be determined.
- The left and right subtrees can be built recursively.
- One or more base cases are needed to stop the recursion.

Discussion of these observations then leads to a pseudocode algorithm, such as:

```
- If the traversals are empty, return null
- Create the root using the first node in the preorder traversal
- If the traversal has a single node, return the root
- Find the preorder and inorder traversals of the left subtree
- Use recursion to build the left subtree
- Find the preorder and inorder traversals of the right subtree
- Use recursion to build the right subtree
- Return the root
```

2.3 Application – the code magnet microlab

The Head First Java book [10] presents exercises for students using a "code magnets" approach in a static format. We have adapted this approach as a way to allow the student to enter program code for methods using the "drag and drop" approach from logical microlabs to create methods that can be compiled and tested. Figure 4 shows the list of magnets available to create the method that will build a general binary tree (GBT) from the preorder and inorder traversals.



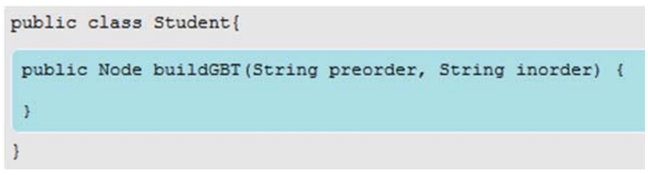
Build GBT from Traversals

Arrange the magnets to build a method that will reconstruct a tree given the preorder and in order traversal of that tree.

```
return null;
return root;
return root;
if (preorder.length() == 0) {
}
if (inorder.length() == 1) {
}
root.setLeft(buildGBT(leftPreorder, leftInorder));
String leftInorder = inorder.substring(0,
inorder.indexOf(rootValue));
String rightInorder =
inorder.substring(inorder.indexOf(rootValue)+1);
root.setRight(buildGBT(rightPreorder,
rightInorder));
Node root = new Node(rootValue);
String leftPreorder =
preorder.substring(1,
leftInorder.length()+1);
String rightPreorder =
preorder.substring(leftInorder.length()+1);
char rootValue = preorder.charAt(0);
```

Figure 4: Code Magnets for the buildGBT Method

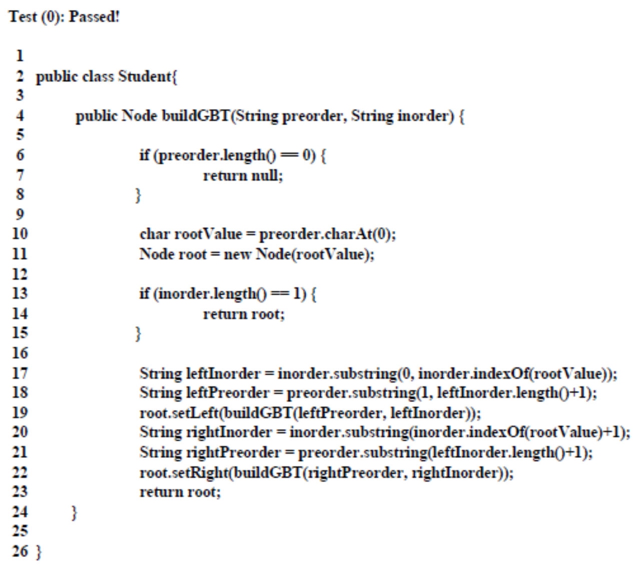
The student drags and drops these magnets into an initially empty method, shown in Figure 5.



```
public class Student{

 public Node buildGBT(String preorder, String inorder) {

 }

}
```

Figure 5: The Empty buildGBT Method

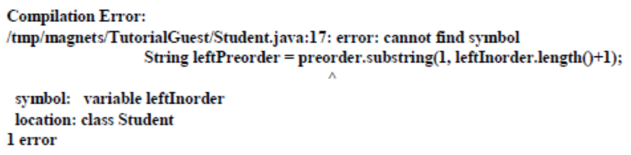
After students place the magnets as desired, they press the Finalize button to compile their code and, if successful, perform unit testing. For the problem shown, the magnets represent actual Java code that is assembled into a program and tested. The feedback for a correct solution for the buildGBT method is shown in Figure 6. In subsequent figures we will show feedback for solutions that do not compile or do not run as intended.



```
Test (0): Passed!


public class Student{

        public Node buildGBT(String preorder, String inorder) {

                if (preorder.length() == 0) {
                        return null;
                }

                char rootValue = preorder.charAt(0);
                Node root = new Node(rootValue);

                if (inorder.length() == 1) {
                        return root;
                }

                String leftInorder = inorder.substring(0, inorder.indexOf(rootValue));
                String leftPreorder = preorder.substring(1, leftInorder.length()+1);
                root.setLeft(buildGBT(leftPreorder, leftInorder));
                String rightInorder = inorder.substring(inorder.indexOf(rootValue)+1);
                String rightPreorder = preorder.substring(leftInorder.length()+1);
                root.setRight(buildGBT(rightPreorder, rightInorder));
                return root;
        }

}
```

Figure 6: A Correct Solution for buildGBT

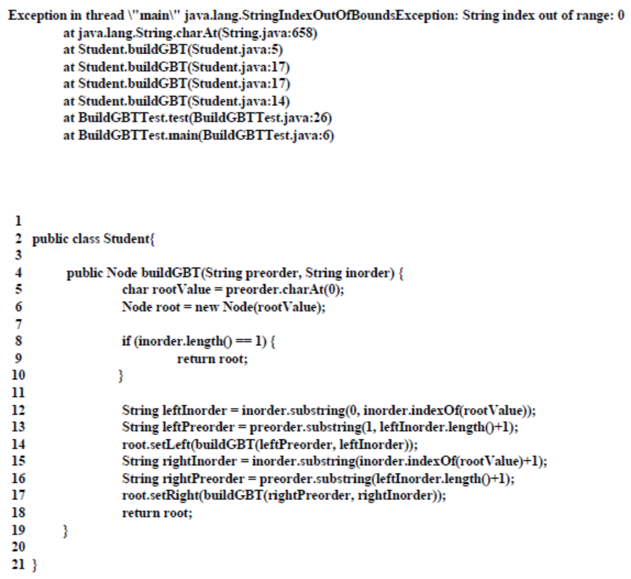
Notice that the length of the inorder traversal of the left subtree is used to construct the left preorder traversal. If the student places these code magnets in the incorrect order, there will be a compile error as shown in Figure 7.



```
Compilation Error:
/tmp/magnets/TutorialGuest/Student.java:17: error: cannot find symbol
                        String leftPreorder = preorder.substring(1, leftInorder.length()+1);
                                                                    ^
  symbol:   variable leftInorder
  location: class Student
1 error
```

Figure 7: A Method with a Compile Error

It is important that the test program includes limiting cases. For the buildGBT method having empty traversals is a limiting case. If the student does not include the magnets for this base case, as shown in Figure 8, the code compiles but produces a runtime error when the unit test with empty traversals is performed.



```
Exception in thread \"main\" java.lang.StringIndexOutOfBoundsException: String index out of range: 0
        at java.lang.String.charAt(String.java:658)
        at Student.buildGBT(Student.java:5)
        at Student.buildGBT(Student.java:17)
        at Student.buildGBT(Student.java:17)
        at Student.buildGBT(Student.java:14)
        at BuildGBTTest.test(BuildGBTTest.java:26)
        at BuildGBTTest.main(BuildGBTTest.java:6)


public class Student{

        public Node buildGBT(String preorder, String inorder) {
                char rootValue = preorder.charAt(0);
                Node root = new Node(rootValue);

                if (inorder.length() == 1) {
                        return root;
                }

                String leftInorder = inorder.substring(0, inorder.indexOf(rootValue));
                String leftPreorder = preorder.substring(1, leftInorder.length()+1);
                root.setLeft(buildGBT(leftPreorder, leftInorder));
                String rightInorder = inorder.substring(inorder.indexOf(rootValue)+1);
                String rightPreorder = preorder.substring(leftInorder.length()+1);
                root.setRight(buildGBT(rightPreorder, rightInorder));
                return root;
        }

}
```

Figure 8: A Method with Runtime Errors

In the code magnet lab illustrated above, every given magnet is used to create the correct method. Often it is pedagogically useful to include additional magnets, say one with a < comparison and another with a <= comparison, so that the student must decide which magnet is logically correct. Problems with alternative magnets are more challenging and require more time for students to solve. The instructor can choose whether to include alternative magnets or not. Alternative magnets work well for simple algorithms often found in a CS1 or CS2 course. For complex algorithms, such as the buildGBT, alternative magnets, such as modifying parameters in the substring operation, may appear to be "tricky", slow down the solution process, and not contribute to the main purpose of the microlab (the use of recursion with smaller traversals and including the base cases).

Code magnet labs are not intended to replace programming assignments. Rather they replace explanations of code during lecture with an activity that forces each student to construct the algorithm interactively. Follow up activities can be the basis of programming assignments. For example for the build tree activities described above, students may be asked to write a complete program, including testing, to build a binary tree given the inorder and postorder traversals. Readers interested in trying a variety of logical or code magnet microlabs can visit [11].

We have tested the code magnets approach with students working on tablets, laptops, and desktop computers. Figure 9 shows a student working on a 7" Google Nexus tablet.

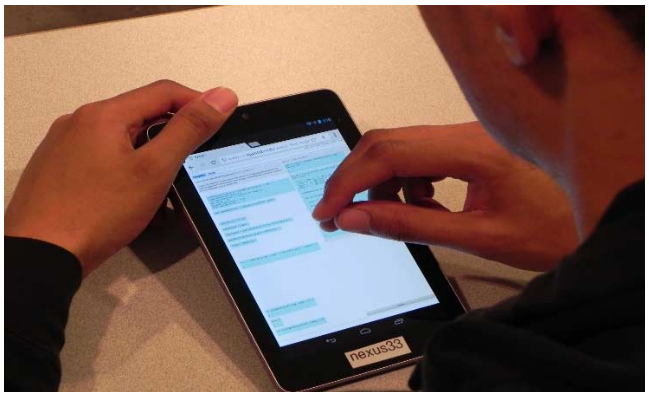


Figure 9: Solving a Code Magnet Problem on a 7" Tablet

3. TESTING WITH TABLETS

3.1 Results from Appalachian State

All testing of microlabs in the 2012-2013 academic year was performed on 7" tablets, as seen in Figure 10. We selected the Nexus 7 tablet because it was affordable and could be easily distributed to students at the start of class. It also provided a good test of the lower limits on screen size where students are still able to complete lab activities. Figure 11 shows use of tablets in a large lecture hall.

Figure 10: Students Using Microlabs on the Google Nexus 7

Figure 11: Using Tablets in a Large Lecture Hall

Testing was performed in multiple sections of three courses we call CS1, CS2, and CS3:

- CS1: Introduction to programming using Java and BlueJ
- CS2: Finishes leftover topics in Java (e.g., GUI, file I/O, exceptions, threads), introduces linear data structures (lists, stacks, queues), covers searching and sorting
- CS3: Completes data structures (dictionaries, binary trees, search trees, balanced search trees, heaps, graphs) and introduces advanced algorithms

Code magnets were used at all three levels; logical microlabs were only used in the CS3 course.

Four sections of the CS1 class participated to test code magnet labs taught by three different instructors (sections 101 and 103 had the same instructor). Usage varied with instructor; section 104 used six labs, 101 and 103 used nine labs, and 102 used thirteen labs. As shown in Table 1, the success rate for 647 attempted labs was 89.7% correctly solved by the start of the next class period.

Table 1 – Results for CS1

Topic	# sections	# students	% correct
Arrays: find max	3	57	82.5
Arrays: find average	3	50	98.0
Arrays: index of max	3	53	77.4
Arrays: swap min max	3	39	71.8
ArrayLists:Ins/Remove	2	36	72.2
ArrayList: Add	2	38	97.4
ArrayList: ArithSeries	2	33	84.8
Strings DoubleLetter	4	76	97.4
Strings: DoubleWord	4	73	98.6
Strings: Count Occur.	4	72	98.6
Strings: Palendrome	1	15	73.3
HashMaps: Insert	2	39	97.4
HashMaps: Remove	2	36	97.4
for: sumOddNumbers	1	16	87.5
for: isPerfectSquare	1	14	64.3
TOTAL		**647**	**89.7**

There were two sections of CS2 taught by two different instructors. Each section used the same seven code magnet labs (Table 2). These labs were more challenging than the CS1 labs. The overall success rate was 81.4%. The first lab, a method to check the balance of parentheses in an expression using a stack, caused the most difficulty with a success rate of 56.7%. In general we have found performance improves as students become more familiar with microlabs.

Table 2 – Results for CS2

Lab Description	# students	Overall % Correct
Stacks: Balance Checker	30	56.7
Stacks: Postfix Helper	32	96.9
Stacks: Postfix Evaluation	28	71.4
Big Integer: Add	38	71.1
Big Integer: Digit Multiply	33	90.9
Big Integer: Multiply	32	90.6
Queues: Josephus	33	90.9
TOTAL	**226**	**81.4**

CS3 students used both logical microlabs (30 different labs covering 11 topics) and code magnet microlabs (7 distinct labs). The logical labs were grouped into clusters of 2 to 4 problems of the same type (Table 3). Earlier use of these labs in the previous semester indicated that some students used a "guess and check" strategy where these students rapidly submitted alternative solutions until they stumbled onto the correct solution. By varying the grade based on the number of submissions (100% for 1-5, 90% for 6-10, and 80% for more than 10) we found the average number of submissions dropped dramatically to slightly less than two attempts until a correct solution was found. The average score for the logical microlabs was 95.7% (287.2/300).

Table 3 – Results for Logical Microlabs in CS3

CS3 logical labs	#	Max	Avg
Insert into a BST	2	20	20.0
Build a BST from postorder	3	30	29.6
Build GBT from traversals	3	30	29.6
Hashing - linear probe	2	20	20.0
Hashing - quadratic probe	2	20	19.8
Insert into a heap	4	40	36.5
Delete from a heap	2	20	17.9
Build a heap	4	40	37.1
Heapsort	2	20	16.7
Min Span Tree - Prim	3	30	30.0
Min Span Tree - Kruskal	3	30	30.0
TOTAL	**30**	**300**	**287.2**

Table 4 shows the results for the code magnet microlabs used in CS3. The average of 93.6% (65.5/70) is substantially higher than other graded activities in the course.

Table 4 – Results for Code Magnet Microlabs in CS3

Lab Description	Max	Avg
Linear search - iterative	10	10.0
Linear search - recursive	10	9.8
Binary search - iterative	10	7.3
Binary search - recursive	10	8.8
Insert nodes into a BST	10	9.9
Build BST from Postorder	10	9.9
Build GBT from traversals	10	9.9
TOTAL	**70**	**65.5**

Long term retention was measured by asking similar problems on the course exams. The microlab exam problems required transfer to a new domain, such as "Given the postorder traversal 1 4 7 6 3 13 14 10 8 draw the original binary search tree." These problems were mid-level in difficutly; there were many easier problems (multiple choice, fill in the blank, short answer) and one or two more challenging problems (write the code for a method from scratch). Table 5 summarizes these results. The overall average of 82.6% (72.7/88) is lower than the microlab performance indicated in Tables 3-4; however, students in an exam setting only have one attempt to solve the problem and receive no feedback. We are encouraged because this 82.6% result represented much better performance than other questions on the exam; the averages for the other questions were 69.7% for midterm 1, 72.4% for midterm 2, and 67.2% for the final exam.

Table 5 – Long Term Retention in CS3

Retention on exams	Exam	Max	Avg
Hashing (2), write iterative binary search code	mid 1	15	11.8
Insert BST, Build BST & GBT, insert heap, heapify, heapsort	mid 2	30	22.7
Traversals, heapsort, hash tables, MST, build binary trees	final	43	38.2
TOTAL		**88**	**72.7**

3.2 Appalachian State Focus Groups

Amy Germuth of EvalWorks visited with students in the four CS1 classes, the two CS2 classes, and the CS3 class. She actually observed students using microlabs in two classes (the HashMaps code magnet labs in CS1 and the minimum spanning tree logic microlabs in CS3). Also, she met with each of the instructors. Her evaluation is presented below verbatim.

> ***There is a direct relationship between exposure to Microlabs and how positively students view them.*** Students who had been exposed to the greatest number of Microlabs found them the most useful, liked them the best, and had much fewer complaints about having to log in, the interface, or how much time they took to complete.
>
> ***Students noted multiple positive aspects of Microlabs***, including that they provide opportunities for practice; are hands-on; provide feedback; engage students in problem-solving; and give the form of coding while allowing students to see the "bigger picture". Students commented that Microlabs "helped me figure out the logic behind the code without having to worry about syntax", "reinforce concepts that we are learning", and "helps you recognize bugs in other people's code". Multiple students suggested that more challenging Microlabs were needed and that instructors should implement a "guessing" penalty so students actually put effort into finding solutions.

3.3 Results from UNCG and Stony Brook

In the Spring 2012 semester 22 students in a data structures class at UNCG completed the logical and the programming microlabs for the binary tree topics. There were no magnet microlabs and no tablets involved in this testing. The results are shown in Table 6. These are comparable with more recent results at other schools. It should be noted the scores for programming microlabs where students had to type in code were lower than the corresponding magnet microlabs used in later testing.

Two groups of students participated in a control group – experimental group study at SUNY Stony Brook. In the Spring 2012 semester the instructor used traditional lecture to teach 26 students how to traverse binary trees, how to build a binary search tree (BST) from a postorder traversal, and how to build a general binary tree (GBT) from the preorder and inorder traversals. These topics were tested using three exam questions. A similar group of 25 students was taught the same topics in the Fall 2012 semester using microlabs; this group was given the same set of exam questions. As seen in Table 7, there was an 8 to 10% gain in performance by the group that used the microlabs. When a t-test was applied to these two groups, the result was statistically significant (p=0.038). There was no significant difference in performance between the groups on the remaining test problems.

Table 6: Results at UNCG

UNC Greensboro	% Correct (logical)	% Correct (program)
Traversals	94.1	90.7
BST Insert	89.9	82.1
Build BST	81.2	78.8
Build GBT	79.1	74.5

Table 7 – Results from SUNY Stony Brook

SUNY Stony Brook	traversals	Build BST	Build GBT
Spring 2012	88%	75%	89%
Fall 2012	97%	85%	97%
Gain/Loss	+9%	+10%	+8%

4. CURRENT DEVELOPMENTS

Our microlabs are developed using Google Web Toolkit (GWT) as reported in [5]. Most instructors don't have time to learn GWT to develop new labs from scratch so we are modifying existing labs so instructors can modify content. For example, for the binary tree activities, the instructor can graphically enter the desired tree, either as input (as in traversals) or as the result (as in building a tree from the preorder and inorder traversals). For the logical microlabs dealing with sorting, the instructor can specify the data sets for the sorts. Similar modifications have been made to other logical labs.

Code magnet microlabs are currently implemented for Java. When the instructor designs a code magnet lab there are two levels for each magnet: what appears on the face of the magnet and the code that is generated for compiling and testing. This allows the magnet to appear as psuedocode, such as "Create the root using the first node in the preorder traversal" while the generated code is actual Java code (char rootValue = preorder.charAt(0); Node root = new Node(rootValue);).

We are currently expanding the user interface to allow students to select simple expressions as found in the control of while loops, for loops, and if statements. For example, the code magnet for a for loop will be "for (<<initialization>>; <<termination>>; <<continuation>>) { … }" where each item will be selected from a pull-down menu of possible choices as specified by the problem designer for the given context.

5. CONCLUSIONS

The microlab learning cycle is based on well-established learning theory with three stages of instruction: exploration, invention and application. The logical microlabs are an exploratory activity when the students are asked to solve conceptual problems with a minimum of guidance. Logical microlabs work well on tablets and are the most popular type of microlab. The latest versions can be modified by instructors without having to learn GWT. During the invention phase the instructor guides students in the development of an algorithm to solve the problem introduced in the logical microlab. Then the application phase focuses on using code magnets to construct, compile and test the algorithm using Java program code.

Testing for the last year has focused on tablets as the delivery mechanism, but laptops and desktops can also be used for delivery. We have used microlabs during lecture but they can also be used during closed labs or outside of class. Students have found microlabs to be valuable review tools when studying for exams. A control group – experimental group model was used at SUNY Stony Brook and there was found to be learning gains of 8-10% with a statistical significance of p = 0.038. Similar results are reported at Appalachian State when the success on exam problems based on microlabs was compared with success on other exam problems. We have just received a new NSF grant for the next two years to help disseminate the microlab approach to computer science educators at other universities

6. ACKNOWLEDGMENTS

This work has been supported in part by two active NSF grants: Developing Software and Methodologies for eBook/Browsers to Enhance Learning (DUE1044572) and Transforming Computer Science Education with Microlabs (DUE1122752). Continuing support for the 2013-14 and 2014-15 academic years is provided by Collaborative Research: Innovative Active Learning Using Tablets (DUE 1323178).

7. REFERENCES

[1] Alice Educational Software Website http://www.alice.org/

[2] K. Bowles, A CS1 course based on Stand-Alone Microcomputers, SIGCSE Bulletin, Vol. 9, No. 1, February 1978, 125-127

[3] S. Edwards, M. Perez-Quinones, Web-CAT: automatically grading programming assignments, ITiCSE 2000, ACM Press, page 328

[4] R. Fuller (editor), "A Love of Discovery: Science Education – The Second Career of Robert Karplus", Kluwer Academic/Plenum Publishers, 2002, chapter 4

[5] B. Kurtz, J. Fenwick, and P. Meznar, "Developing Microlabs Using Google Web Toolkit," Proceedings of the 43rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '12), 2012.

[6] B. Kurtz, J. Fenwick Jr., C. Ellsworth, "The Quiver System," SIGCSE Technical Symposium on Computer Science Education, Norfolk, February 2004.

[7] B. Kurtz, H. Pfeiffer, "Developing Programming Quizzes to Support Instruction in Abstract Data Types", SIGCSE Bulletin, Vol. 21, No. 1, February 1989, pp. 66-70

[8] MIT Center for Mobile Learning, "Welcome to MIT App Inventor." Retrieved Aug 25, 2013 from http://appinventor.mit.edu

[9] Scratch Educational Software Website http://scratch.mit.edu/

[10] K. Sierra, B. Bates, Head First Java, O'Reilly Media; 2nd edition, February 9, 2005

[11] Web Automated Grading System, *http://cs.appstate.edu/wags/guest*.

[12] A. Zeller, Making Students Read and Review Code, ITiCSE 2000, ACM Press, pp. 89-9