

Solving the CS1/CS2 Lab Dilemma: Students as Presenters in CS1/CS2 Laboratories

Kay A. Robbins, Catherine Sauls Key, Keith Dickinson, John Montgomery

Division of Computer Science

University of Texas at San Antonio

San Antonio, TX 78249

{krobbins, key, kdickins, jmontgom}@cs.utsa.edu

Abstract

In our quest to modernize our CS1/CS2 curriculum, we ran into several problems in the effective delivery of the courses and their associated laboratories. We have developed a teaching model in which students become the presenters for the hands-on laboratories. In order for this approach to be effective, the laboratories must be reused from semester to semester, so that student presenters are truly knowledgeable. The student-presenter model also requires more detailed supporting material and a rethinking of course grading policies.

1 The CS1/CS Laboratory Dilemma

Computer Science departments across the country are modifying decades-old approaches to their introductory courses to embrace a *paradigm shift* in the software industry that reflects the visual, object-oriented, event-driven, multithreaded, distributed nature of software [5]. The debated approaches include whether objects should be introduced early, whether to emphasize use of existing objects or design of objects from scratch [2,10] and whether graphics and event-driven execution should be incorporated [11,13]. The new crop of CS1/CS2 textbooks [7,9] reflects myriad variations on these choices, with many supporting an optional *graphics* thread that may be omitted if the instructor chooses. Another issue is whether or not to introduce a graphical development system early on or stick with a text editor and command-line execution [12]. (See also references [3] and [6].)

While these curricular issues are crucial and often hotly debated, a more immediate problem for many departments is finding people with the skills needed to make the transition. CS1/CS2, by nature, are relatively large enrollment courses that act as a conduit of students into the major. Regardless of whether the course is offered in a large lecture with small laboratory sections or in small to moderate size integrated lecture/lab sections, a significant number of skilled teachers are required to staff these courses. Often the laboratories and small sections are staffed by teaching assistants—graduate students who most likely have never taught, may not be familiar with the programming language or design approaches, may not have a polished command of English, and will not be around for very long.

Because of the complexity of these new paradigms, it is nearly impossible to teach this material without significant hands-on exposure. An effective laboratory experience is a key to student success. In this paper we describe an instructional model and supporting curricular materials that we have used to overcome the logistic difficulties in making a paradigm shift in CS1/CS2.

2 Students as Presenters, an Instructional Model

We began our conversion to an object-oriented, Java-based course in the Fall of 1999. Prior to that time our CS1/CS2 courses were taught in C using a text editor and command-line compilation. During the 1-hour laboratory associated with the 3-hour lecture, a TA answered questions and used an overhead projector to present material. Student surveys and the significant drop rate indicated that the greatest weakness was the lack of a hands-on laboratory.

For the new version of the course, we made a commitment to an objects-from-day-one approach. We also sought to provide a more hands-on, collaborative environment that would attract majors and keep them involved in the program. While the conversion has not been without problems, we believe that we are converging on a teaching model that can accomplish these goals within the

constraints imposed by our institution and might be successfully adapted for other institutions.

The teaching model is based three ideas:

1. Reusable laboratories.
2. Student presenters to support teaching assistants.
3. Close integration of laboratory and lectures.

2.1 Reusable Laboratories

If software and software designs can be reused [1], why not teaching materials? The advantages of reuse are obvious—it's less work, the material can be improved over time as instructors get feedback and observe use, and the program builds up a cadre of students who are familiar with the exercises and are capable of providing assistance to their peers. The case against reuse is also clear—students copy.

We devised a grading policy in which half of the laboratory portion of their grade (20% of the total grade) was based on the standardized, reusable laboratories. Another 20% of the total grade was based on individual projects (4 during the semester) that changed every semester and on which students could receive no outside help.

For each course we developed 12 reusable laboratories that were covered roughly one per week during a 50-minute in-class laboratory. An example of a laboratory and supporting teaching materials is presented in the Appendix.

To receive credit for the laboratory, students were required to attend the laboratory class and to hand in a set of deliverables a week later. Students were encouraged to get as much help as they needed on the laboratories. The goal of the in-class laboratory was to give students a thorough understanding of the material and a good start on coding the lab. The concepts needed in the laboratories were covered in lecture prior to the laboratory meeting. Care was taken that the 4 individual projects used skills that were developed in the laboratories and that the relationship of the skills to the projects was obvious to the students. Lectures also referred to the Java classes developed in the laboratories in later examples, so that the students saw the laboratory code several times during the semester.

2.2 Student Presenters

One of the difficulties with the paradigm shift is that more sophisticated material is incorporated into CS1/CS2, while students are probably not getting smarter. When we started the transition, it became clear that quality-control in the teaching of the laboratories was a problem. The additional complexity of the software and systems meant that the TAs often spent a considerable portion of their teaching time handling computer problems. The TAs, were for the most part, unfamiliar with Java and the labs required significant preparation on their part. It was too much to ask a TA to also grade the course, so we had to find graders who knew the material.

After commiserating that good students taking the class could do a better job of presenting the laboratory material, we decided to change our teaching model. We hired a student presenter for each laboratory session. The presenter took the students through the laboratory, following a script. The TA handled logistics, account problems, attendance and grading. We also hired some of the best students who were taking or had just completed CS2 as tutors. The tutors staffed the laboratory approximately 14 hours per day/7 days per week. These tutors were available to sit with students who were having trouble completing a laboratory. Tutors handled account problems and would re-explain lecture material.

2.3 Integration of Laboratories with the Lecture

An advantage of an effective laboratory experience is that it frees up lecture time to do more design. We made an effort to discuss the design aspects of each laboratory during the week before the laboratory occurred. We also made sure that we reused the Java classes written in the laboratories in examples throughout the course.

3 Instructional Materials

In the first laboratory, which is covered during the first two weeks of the semester, students receive their accounts, copy a simple hello-world program, learn to send email and create a personal web page. Because of adds, drops and general confusion, we can not assume that everyone will be ready with an account and system familiarity until the end of the second week of class. The sample laboratory shown in Appendix A is covered in the third week of class.

All laboratories have the same format. They begin with an *Objectives* section that briefly states the purpose of the laboratory. These objectives can be related to topics in the syllabus. The next section, entitled *Prior to the Laboratory*, indicates how the students are expected to prepare for the laboratory session. Usually some sort of supporting worksheets (see Appendix A.2) are available. This section may indicate parts of the text to read or be keyed to lecture notes available on the web. The laboratory then explicitly details the *Deliverables*. Following this section are a detailed specification of the problem and *Study Questions* that must be completed after the laboratory session.

A key to the effective presentation is for the lecture instructors to work with the student presenters on how they should teach the material. We also developed teaching scripts for each laboratory (see Appendix A.3). These scripts are revised and polished as we receive feedback from the TAs and presenters.

2 Student Reaction

A survey was administered to students at the end of the CS2 course. Students were asked to compare their experiences in the CS2 lab with their experiences in other

courses in previous semesters. (The students who were surveyed had taken CS1 without presenters.) Students overwhelmingly said that the presenter was able to help them with difficult concepts, came prepared for the class, and gave explanations that were clear and easily understood. They also mentioned that with two instructors in the classroom, one could walk around the room helping students on an individual basis, while the other instructor could continue the class without disruption. The improved quality and quantity of the student tutors available throughout the day was the aspect mentioned most often by the students in this survey.

3 Discussion

The student presenter teaching model for the laboratories has many benefits, both for the students taking the course and for the presenters themselves. The presenters learn to present technical material in a controlled environment and the enrolled students are clearly getting a better course. We are able to reach students on the edge, and students generally feel the program is more hospitable. As employees, the student tutors and presenters have a vested interest in helping other students in the program succeed. We also found that we were able to use more lecture time for design and problem-solving and less time for syntax and system issues. The presenter format forced standardization of the labs among all of the lecture sections, improving quality control and defining a specific set of baseline skills for the course.

These benefits don't come for free. We spend about \$12,000 per long semester to fund the tutor/presenter program. While this is not a large amount of money, it is not an expense that is typically covered in funding formulas at state institutions such as our own. We have been fortunate to receive NASA funding for our pilot program, and are seeking long-term funding from local industry. This teaching model also requires a significant amount of up-front curriculum development. The labs and scripts must be written for the presenter format. Care must be taken to balance what will be done by the presenter with the students just copying, what will be done by the students with the presenter and TA giving individual help, and what will be done by the student after the laboratory.

In addition to the curriculum development, the faculty members in charge of the course have to spend time with the presenters to sure that they understand the material and can explain it effectively. This activity is directly related to the presenter's learning, in contrast to that of the graduate teaching assistant, whose program of study may be unrelated to the course material.

The curriculum materials that we have developed for CS1/CS2 in the presenter format are available through <http://vip.cs.utsa.edu/NASACurriculum/>.

Acknowledgments

This work was made possible by a NASA Grant MURED PTNR-E. We also wish to thank Charlie Collins for his work in setting up the laboratory.

References

- [1] Astrachan, O., Berry, G., Cox, L., and Matchener, G. Design patterns: An essential component of CS curricula. *29th SIGCSE Technical Symposium on Computer Science Education* (1998) 153-160.
- [2] Astrachan, O. and Rodger, S. Animation, visualization and interaction in CS-1 assignments. *29th SIGCSE Technical Symposium on Computer Science Education* (1998) 317-321.
- [3] Bergin, J., Naps, T., Bland, C., Hartley, S., Holliday, M., Lawhead, P., Lewis, J., McNally, M., Nevison, C., Ng, C., Pothering, G. and Terasvirta, T. Java resources for computer science instruction, *ITiCSE Working Group Papers* (1998) 30(4):18b-38b.
- [4] Boroni, C. M., Goosey, F., Grinder, M. and Rockford, J. A Paradigm Shift! The internet, the web, browsers, Java, and the future of computer science education. *29th SIGCSE Technical Symposium on Computer Science Education* (1998) 145-152.
- [5] Culwin, F. Object imperatives! *30th SIGCSE Technical Symposium on Computer Science Education* (1999) 31-36.
- [6] Knox, D. The Computer Science Teaching Center. *SIGCSE Bulletin* (1999) 31(2):22-23.
- [7] Koffman, E. and Wolz, U. *Problem Solving with Java*. Addison Wesley (1999).
- [8] Kolling, M. and Rosenberg, J. Objects first with Java and BlueJ. *31st SIGCSE Technical Symposium on Computer Science Education* (2000) 429.
- [9] Lewis, J. and Loftus, W. *Java Software Solutions: Foundations of Program Design*. Addison Wesley (2000).
- [10] Long, T., Weide, B., Bucci, P. and Sitaraman, M., Client view first: An exodus from implementation-biased teaching. *30th SIGCSE Technical Symposium on Computer Science Education* (1999) 136-140.
- [11] Regis, S. Conservatively radical Java in CS1. *31st SIGCSE Technical Symposium on Computer Science Education* (2000) 85-89.
- [12] Warford, J. BlackBox: A new object-oriented framework for CS1/CS2. *30th SIGCSE Technical Symposium on Computer Science Education* (1999) 271-275.
- [13] Woodworth, P. and Dann, W. Integrating console and event-driven models into CS1. *30th SIGCSE Technical Symposium on Computer Science Education* (1999) 132-135.

Appendix A: Sample Instructional Materials

A.1 A Laboratory Handout

Laboratory 2: A Coin Counter

Objectives:

- Write a simple class.
- Become familiar with basic object terminology.
- Learn to identify constructors, accessors and modifiers.
- Write client code to instantiate objects.
- Use counters.

Prior to the Laboratory:

- Read through the laboratory.
- Try to fill in the worksheet for the `CoinCounter` class.
- Try to fill in the worksheet for the `CoinCounterTest` class.

Hand-in Requirements (Deliverables):

- **Attempted worksheets:** `CoinCounter`, `CoinCounterTest`
- **Hardcopy:** `CoinCounter`, `CoinCounterTest`
- **Study questions**

Overview:

Grocery stores and banks have coin exchange machines that allow customers to deposit unsorted coins and receive the equivalent amount of cash in return. A coin exchange machine has two components: a coin sorter that separates the coins by type and a coin counter, which given the numbers of coins of each type, figures out the cash equivalent. In this laboratory you will develop a `CoinCounter` class to represent the second component of the coin exchange machine. You will also develop a test program called `CoinCounterTest` to test your `CoinCounter` class.

Objects of type `CoinCounter` allow the user to make deposits, get the total value of the money deposited, and print the totals of the each of the coins along with the total value of the deposits.

1. The *fields* of a class are internal variables for the holding information that the object keeps track of. `CoinCounter` objects keep track of how many of each type of coin they have (pennies, nickels, dimes and quarters). The fields are initialized to zero when a `CoinCounter` object is created (in the *constructor*). The fields are *private*.
2. The *public methods* are the actions that the outside world can ask the object to take. The *accessor methods* provide information about the object, but don't change it, while the *modifier methods* cause the object to change in some way.

The `CoinCounter` class has accessor methods:

- `getTotalDeposit` - returns amount deposited in dollars.
- `print` - outputs in readable form the numbers of the different types of coins and the total value of the deposits.
- `toString` - returns information about `CoinCounter` as a `String`

The `CoinCounter` class has *modifier methods*:

- `clear` - set all of the totals to zero.
- `deposit` - add additional coins to the deposit.

Details:

• Setup:

- Create a new project called `CoinCounterTest` and add a new class file called `CoinCounter.java` to the project.
- Save `CoinCounter.java` and `CoinCounterTest.java` from the course web site in the project directory.
- Build and run to make sure that your setup is correct.

• Writing and testing the print method

- Implement the `print` method of `CoinCounter`.
- Write client code (in the `CoinCounterTest.java` file) to create a `CoinCounter` object called `aCounter`.
- Write client code to call the `print` method of `aCounter`.

• Writing and testing the deposit method

- Implement the `deposit` method of `CoinCounter`.
- Write client code to deposit 5 quarters, 2 dimes, 1 nickel, and 7 pennies in `aCounter`. Output `aCounter`.
- Write client code to put the value of the coins deposited so far into a `double` variable called `totalValue`. Then print `totalValue` with an identifying label.
- Write client code to deposit an additional 3 quarters, 8 dimes, and 4 nickels in `aCounter`. Again find out and print the value of the total deposits.

• Implementing the rest of CoinCounter:

- Implement the remaining methods of `CoinCounter`.
- Write client code to create an additional `CoinCounter` object called `bankCounter`. Make some deposits to `bankCounter`.
- Write client code to print `aCounter` and `bankCounter`.
- Write client code to clear `aCounter` and print it.
- Test the remaining methods of `CoinCounter`.

Study Questions:

1. What type of value does `getTotalDeposit` return?
2. Draw a picture of `aCounter` after the first deposit (5 quarters, 2 dimes, 1 nickel and 7 pennies).
3. Suppose `CoinCounter` had a `setQuarters` method that set the number of quarters to a specified amount. Is `setQuarters` a modifier or an accessor method?
4. What accessor methods might you add to `CoinCounter` so that clients could find out the number of each kind of coin?

A.2 Sample Worksheets

```
public class CoinCounterTest {
    public static void main(String args[]) {
        // instantiate a CoinCounter, aCounter

        // print aCounter

        // deposit 5 quarters, 2 dimes, 1 nickel,
        // 7 pennies in aCounter, print aCounter

        // set totalValue to the current value of the
        // total deposit of aCounter and print

        // deposit 3 quarters, 8 dimes and 4 nickels
        // in aCounter and output the total value.

        // create a CoinCounter called bankCounter

        // remaining test code...
    }
}
```

```

public class CoinCounter {
    // constants giving the coin values in $
    static final double QUARTER_VALUE = 0.25;
                                     // <==
                                     // <==
                                     // <==

    // fields contain the object's internal state
    private int myQuarters; // total quarters
                          // total dimes <==
                          // total nickel <==
                          // total pennies <==

    // constructor (initialize object on creation)
    public CoinCounter( ) {
        myQuarters = 0; // no quarters
                       // no dimes <==
                       // no nickels <==
                       // no pennies <==
    }

    // accessors (methods don't change object)
    public double getTotalDeposit( ) {
        // return total value of coins deposited in $
        return 0; // <==
    }

    public void print( ) {
        // output the coin values in a nice format
        // <==

        // <== toString method

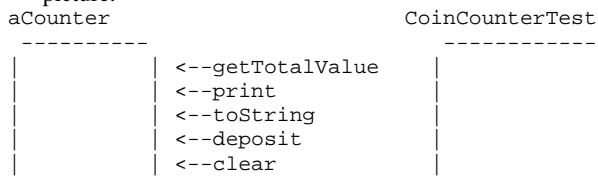
    // modifiers (methods change object's state)
    public void deposit (int quarters, int dimes,
                        int nickels, int pennies) {
        // update the totals to reflect this deposit
        // <==
    }

    // <== clear method
}

```

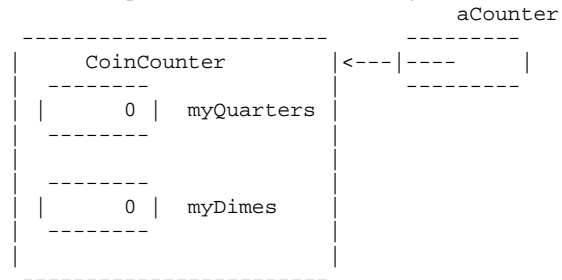
A.3 A Sample Presenter Script

- (Start promptly at the beginning of class even if everyone is not ready. You should have the machine and projector set up before the class period begins.)
- Please take out your handouts. Does anyone need a copy?
- Go through the setup part and give some help to make sure that most students have the code setup. (Use your judgment.)
- In this laboratory we will develop and test the `CoinCounter` class. What are `CoinCounter`'s public methods? Draw a picture:



- What are `CoinCounter`'s fields? In this session we will use only quarters and dimes. You can fill in the nickels and pennies after the lab. Add a field for dimes. (Demonstrate by adding `myDimes`.) What does the word `private` mean? Why is `myDimes` an `int`?

- Where does the code to initialize object go? Add a statement to the constructor to initialize `myDimes` to 0 when a `CoinCounter` is created. (Demonstrate.)
- Build and run the project. (Demonstrate.)
- When I say "add client code", where do I put it? Edit the `CoinCounterTest.java` main program.
- Add code to instantiate a `CoinCounter` object called `aCounter`. (Wait a minute and then demonstrate.) Build and run.
- Here is what happened when we instantiated `aCounter`. (Draw a picture on the board of the object `aCounter`):



- Now ask `aCounter` to print by calling its `print` method. Build and run. (Wait and then demonstrate.) Why didn't anything print out? (ANS: we haven't written the code yet.)
- We will have to edit the definition of `CoinCounter` so that `aCounter` can print. What should be output? Modify the `print` method of `CoinCounter` to output the number of quarters and dimes.
- We also need to print out the value in dollars of the deposit? How do we calculate the value of the deposit? (Write the statement on the board.) Lets define a variable inside of `print` to hold the value. What kind of variable should it be? (`double`) Build and run. (Demonstrate)
- Add the code for the `deposit` method. The `quarters` variable is a *parameter*. Its value is set by the client code to say how many quarters have just been deposited. The value of `quarters` should be added to `myQuarters`. (Demonstrate) Now handle the dimes in the same way.
- Build and run. Add code in the main program to deposit 5 quarters and 2 dimes. Call the `print` method right after `deposit` to make sure it works.
- Now we will write the `getTotalDeposit` method of `CoinCounter`. What kind of value does `getTotalDeposit` return? (Show where the type of return value is specified.)
- Now write client code to test `getTotalDeposit`. Go to your `CoinCounterTest` program. Since `getTotalDeposit` returns a value, we will need to put it somewhere or use it right away. In this case we are going to put the returned value in a variable called `totalValue`. (Demonstrate.)
- Build and run.
- Lets look at the `CoinCounter` again. Notice we have the same code in both `getTotalDeposit` and `print`. Lets rewrite `print` so that it calls `getTotalDeposit`. Notice that we don't put the object name in front of the call. It is a very good idea not to duplicate code for formulas. If there is a change in the way it is computed, you only have to change it in one place. (Demonstrate and ask for questions.)
- Now try to complete the remaining statements of the assignment. You will have to add codes for nickels and pennies. You will also need to write `clear` and `toString`.