# Engaging Multidisciplinary Engineers in an Introduction to Programming Laboratory

**Dr. Ira Hill, University of Florida**

Ira Hill is a faculty member in the Institute for Excellence in Engineering Education at the University of Florida, which focuses on improving large-enrollment, introductory engineering courses. Dr. Hill currently teaches programming for engineers across all majors. His research interests include developing and incorporating engaging demonstrations into the classroom and faculty development. His educational background includes a B.S. in Mechanical Engineering from the University of Pittsburgh and a M.S. and Ph.D. from the University of Florida. He has experience in implementing robotics solutions for biomechanics applications, including a postdoctoral fellowship with the UF Orthopaedics and Sports Medicine Institute.

# Engaging Multidisciplinary Engineers in an Introduction to Programming Laboratory


## Abstract

Engineering students outside of computer science are required to take an introductory course in computer programming in one of several languages (MATLAB, C++, VB.net), including a laboratory component. This provides a unique challenge in engaging a group of multidisciplinary students with different programming backgrounds, especially since the lab is required by some engineering majors but optional for others. The lab had essentially turned into a recitation session with additional lecturing and reviews of homework solutions. Over the last several semesters the college has reevaluated how the lab can be useful to all disciplines, and this paper outlines the curriculum redesign to problem-based learning in a collaborative classroom. Students now work in a space designed for active learning for two periods each week, grouped in teams of six. Their goal is to solve programming challenges that range from programming fundamentals to image processing and manipulating experimental data, which stimulates the interest of all engineering disciplines. Example labs include solving programming interview questions, using image kernels to sharpen digital images, and developing a simple Microsoft Paint application. These challenges correspond to the latest lecture material, forcing students to actively work through the current learning objectives and keep pace with the course. Each lab session has the support of a faculty member and teaching assistants to guide discussions and provide just-in-time teaching. Student feedback and grades have shown students are meeting the desired learning objectives while also enjoying the challenging nature of the problems. Students with no prior programming experience have especially benefited from the new lab format with strong improvements in critical thinking, creativity, and problem solving skills.


## Introduction

Teaching non-computer science majors programming fundamentals has posed several unique challenges in our Introduction to Computer Programming course (COP2271) at the University of Florida. The course traditionally supported several majors and different programming languages through separate sections, including the Fortran and C languages, with a lecture and laboratory component that combined for 3 or 4 credits. Previous faculty members took different approaches about the content and learning objectives of the course, leading to inconsistent learning outcomes for students. This also made it difficult for departments to predict the programming skills their students would have in future classes; invested departments include Aerospace, Biological, Biomedical, Chemical, Electrical, Environmental, Material, Mechanical, and Nuclear.

While changes have been made to the lecture component of COP2271, the laboratory component has been significantly improved. Traditionally, the lab functioned as a recitation led by a teaching assistant who covered previous homework solutions or worked example problems. This was generally not well received by students and a curriculum change was implemented beginning in 2015. It was our belief that students needed to physically code more through guided practice on complex problems to be successful programmers. Programming fundamentals such as variables, if statements, and loops have a relatively easy syntax to memorize but are difficult to creatively apply to practical engineering problems. Felder and Brent confirm this intuition

with several studies that show students need repetitive practice with consistent feedback to develop new skills (1). Simply showing students how to solve a particular problem doesn't guarantee they can apply these concepts on their own. With these ideas in mind, the lab morphed from a traditional recitation to weekly programming challenges solved in a group setting. The laboratory and lecture now focus on the C++ and MATLAB languages with plans to also incorporate Python in future semesters.

This paper details the changes to the laboratory portion of the course to use problem-based learning (PBL) and just-in-time teaching (JiTT) in a collaborative student space and gives examples of the group activities and their relationship to the course learning objectives. The success of the changes is shown through student feedback and survey results.

## Methodology

Students now meet once a week in an active learning space designed with 10 rounded tables holding six students apiece. Each table is equipped with two large monitors and connections for students to share their laptop screens while still viewing notes from the faculty member. Following a brief introduction, students are presented two or three programming challenges focused on the current lecture material. The activities are in-class only and must be completed with their group in the two-hour span of the lab (students that finish early often work on the weekly homework). Students are encouraged to discuss the challenges as a group, leveraging their class notes and online resources. The problems are purposely designed to challenge their understanding of the current material, and typically two undergraduate teaching assistants help the faculty member circulate the room and provide assistance.

**Figure 1: Collaborative learning space for Introduction to Programming Laboratory**

This setup is ideal for problem-based learning (PBL) and just-in-time (JiTT) teaching which are inductive teaching methods that combine interactive mini-lectures and collaborative recitations on problems intended to challenge students. Case studies have shown that students should work through problems and misconceptions within in their groups to arrive at a solution (2). At the discretion of the faculty member, a handful of mini-lectures are given to address

common issues; groups of students can easily move to a nearby white-board to listen and take notes. It is important to emphasize that each student within a group must complete and submit his or her own code. This guarantees that students who depend on group members to arrive at a solution still have to physically go through process of creating and debugging code.

The central idea behind the course redesign was to incorporate active learning, allowing students to collaboratively develop code to various challenging problems. The previous course design combined a passive laboratory experience with a traditional lecture, preventing students from receiving guided practice and providing little motivation to learn. This contradicts educational theory that shows students learn best when shown the usefulness of the material and how it can impact their lives (3). Numerous inductive or experiential learning techniques exist to address these issues, including case-based learning, project-based learning, discovery learning, and more. PBL and JiTT were chosen since these best matched the curriculum goal for students to program more in a collaborative setting. Also, correctly incorporating PBL helps students develop the following skills: 1) flexible knowledge, 2) effective problem solving, and 3) self-directed learning which help promote lifelong learning and compliment the course's learning objectives (4).

The challenges used for PBL are not necessarily unique themselves but rather are tailored to challenge the students at the current point in their learning process. There are numerous online resources with good sample problems, including Nifty Assignments (http://nifty.stanford.edu) and Project Euler (https://projecteuler.net). Code competition websites are also great resources for developing assignments, including Hacker Rank (https://www.hackerrank.com) and Code Chef (https://www.codechef.com). Problems were crafted to meet the following learning objectives over the semester:

- Perform user input and output controlling formatting
- Create, change, and update variables using operators and operands
- Design if statements with compound conditions for decision making
- Implement mathematical algorithms given the underlying formulas
- Utilize while and for loops to control flow of programs
- Work with random numbers for simple games
- Use debugging tools to identify and fix mistakes in code
- Manipulate string data to implement basic encryption techniques
- Manipulate image pixels as three-dimensional data sets
- Develop basic graphical user interfaces (GUI)
- Collect and process data from data acquisition devices

The following table highlights previous activities chosen and the subsequent sections detail three of the activities:

**Table I**
**A sample of the current laboratory activities used throughout the semester for both MATLAB and C++.**

| Example Lab Activities | Programming Concept Covered |
|---|---|
| Math based problems: resistors in parallel compound interest, Taylor Series Approximation | Variables, operators, basic syntax |

| | |
|---|---|
| Google Currency Converter | If statements |
| Closest Fibonacci Number | While loops, recursive equations |
| Simple Games: Game of Pig, Rock-Paper-Scissors | Pseudo-code, random numbers |
| Google GPS Plotter | File manipulation, strings |
| Microsoft Paint Clone | Pixel manipulation |
| Image manipulation with kernels | Multi-dimensional data |
| UPC Barcode Reader | Algorithm Implementation |
| Speed Gait | Data acquisition |

Closest Fibonacci Number

The Fibonacci sequence is a classic computer science problem involving a recursive equation. This gives students practice in implementing mathematical equations in a loop, properly updating variables, and terminating loops appropriately. It is also a good example of recursive programming for students looking for an additional challenge. Students are given a brief overview of the following equation to begin the lab:

$$\boldsymbol{F_n = F_{n-1} + F_{n-2}}$$

First, students are challenged to simply generate the sequence to an arbitrary value of n. Next, students must take any user input and determine if that value is a Fibonacci number. If not, the code must report the closest Fibonacci number (either lower or higher than the original). This extra component forces students to incorporate if statements to make a final decision with their data.

Recursive equations are traditionally difficult for new programmers, especially as they learn the basics of loops. This is a perfect example of the JiTT mini-lecture to help illustrate how to translate the Fibonacci sequence into code. Groups that are doing well are encouraged to continue working while other groups interactively decipher the sequence with the faculty member at the closest white board. Typically this lab resonates with students once they grasp how the individual parts of the equation are simply three variables that rotate values as the sequence progresses.

Microsoft Paint Clone

It is important for new programmers to appreciate the complexity that goes into even the more simplistic programs, such as the free paint program included in Microsoft Windows. While this challenge isn't engineering specific, it does get students thinking about implementing programming solutions for tasks in their field. A difficulty of this lab is to not overwhelm students with material since a graphical user interface (GUI) requires concepts not covered in the lecture. To mitigate this, students are given a reference framework implementing more of the complex graphical components but with blank sections for students to complete.

The framework for the C++ section of the course uses the Simple Fast Multimedia Library (http://www.sfml-dev.org) while the MATLAB section uses the built-in graphical user interface module. These frameworks create a blank window, allow the user to control the mouse and keyboard, and provide a graphical component to draw in (see Appendix A and B for examples). Note that students do not need to necessarily understand the framework to complete their assignment but are encouraged to explore and ask questions. Students are tasked to complete the program so the user can draw with at least 5 different paintbrush sizes (either square or circular) in 5 different colors. Students then draw any school appropriate picture and submit their code and drawing as the assignment. Examples from four different students from last semester are shown in the collage below:

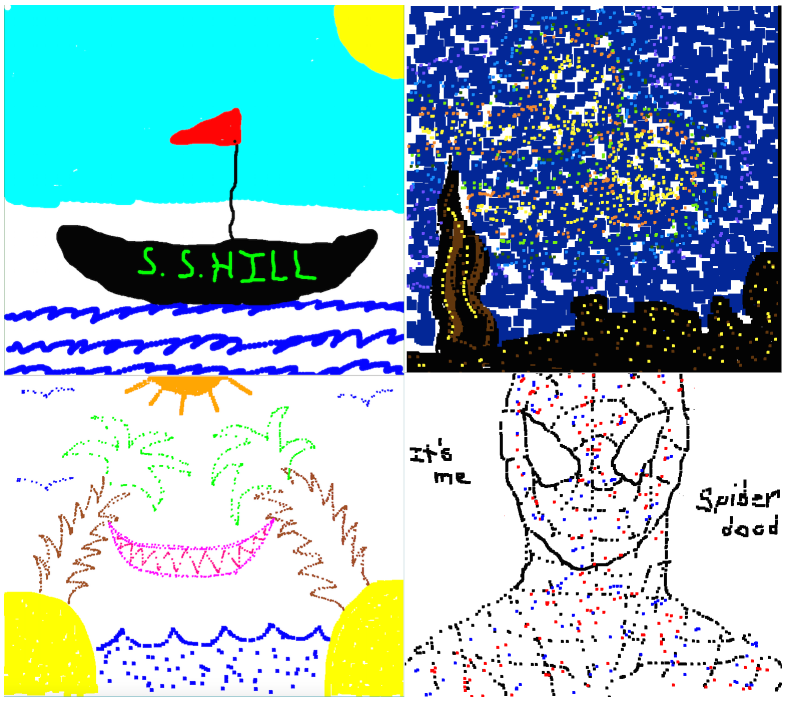


**Figure 2: Collection of student images drawn from their own Microsoft Paint Clone program written in C++ and MATLAB.**

Speed Gait

This lab activity is different from the others in that students don't explicitly write code; instead they reason through a design experiment to develop an apparatus for measuring a person's average gait speed. As a mini-lecture, students are shown various gait-measuring techniques such as the Tekscan Gait Mat and camera-based motion capture, common devices used in biomechanics lab. Each group then details how they would develop an alternative, low-cost system to measure average gait that is more accurate than a human with a stopwatch. Their analysis must include hardware and software components, including pseudo-code of how to program their system. The groups always come up with various creative implementations, including Bluetooth sensors in shoes and proximity sensors in the floor.

After each group shares their ideas, the class builds a laser-based system to complete the activity. This particular system was chosen because the parts are cost effective and easy to

connect with C++ and MATLAB. The main components include two 5-volt lasers, two solar cells, camera tripods, and a USB data acquisition device (any brand is sufficient, including the Arduino or National Instrument devices). The premise is simple; a person walks in front of the first laser tripping a timer, which measures time until the second laser is reached. Speed is calculated given the distance between the lasers, approximating a person's average walking speed. Student volunteers put the pieces together, making decisions such as the optimal distance between lasers and the vertical height above the floor. Furthermore, small gaps are left in the code for students to complete before running the system. Finally, students are encouraged to walk at their normal pace to see how their walking speeds compare to each other.

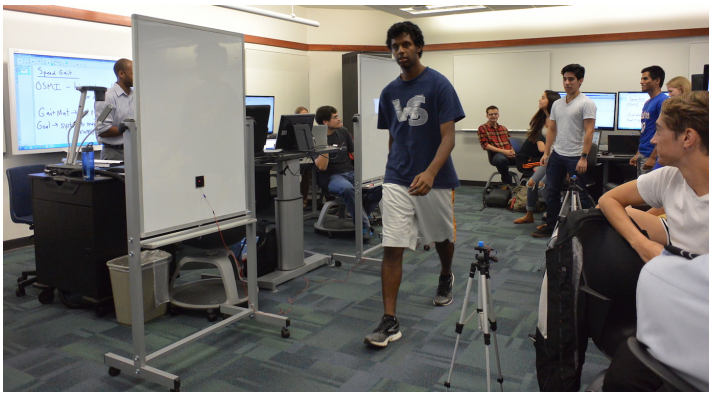

**Figure 3: Students testing their gait speed using a laser-based detection system.**

## Course Assessment

To quantify the success of the changes in lab, students filled out an anonymous six-question survey during the last class, separate from the traditional college-wide course evaluations. Four questions used a five level Likert-scale to quantify how students perceived the new lab format, shown in Figures 4-7. The following two discussion questions were also included, allowing students to provide further context on their opinions:

- What was the most memorable part of lab this semester?
- Do you believe attending lab improved your overall grade in the course? Please explain!

The survey included 94 students with 56 students taking the MATLAB section and 38 in the C++ section. The following graphs show the results of the survey broken down between male and female students (engineering major was not included in the survey but has been added for future work).

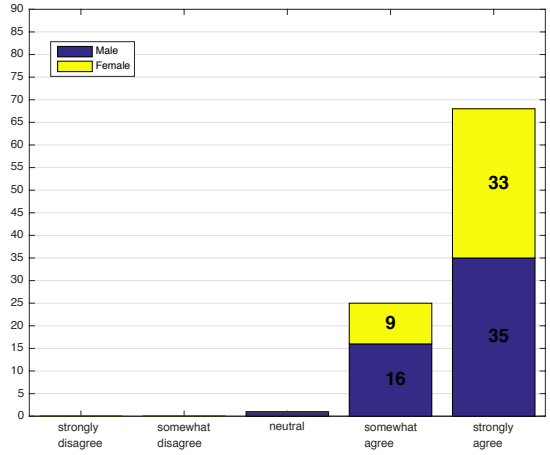


**Figure 4: Did you find the labs interesting?**

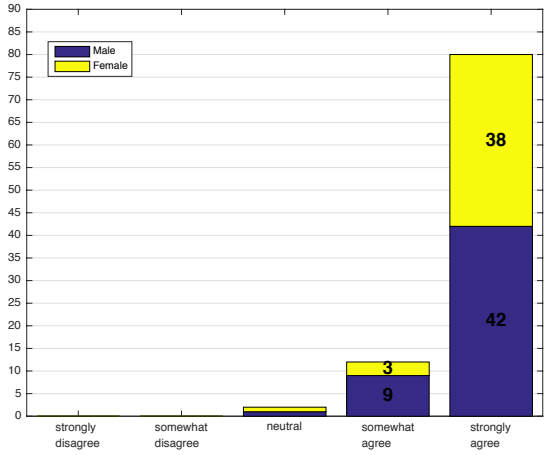


**Figure 5: Would you recommend that future students take the lab?**

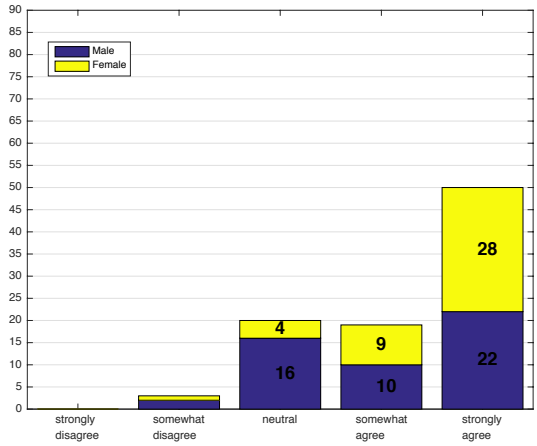


**Figure 6: Attending weekly labs motivated me to attend or watch class regularly**

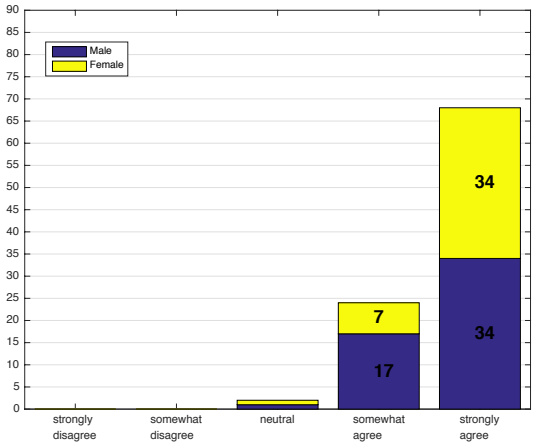


**Figure 7: The labs performed this semester helped me understand the application of course materials to engineering practice and society.**

Several students also left positive feedback for the discussion questions:

- I believe this class really did help my grade because I am a hands-on learner so having lab where I can put everything together and practice really helped!

- Yah!! It allowed for extra practice and forced me to actually apply the things we learned in lecture. It was just really good practice problems and forced me to think more like a programmer.

- Absolutely! By struggling through the labs I was able to figure out the thought process I needed in order to completely the homework assignments. It also helped prepare me for the exams by forcing me to stay up to date on the lectures.

- Yes because it helped strengthen my understanding of the material through the extra practice, and it was being applied to things not covered in normal lecture.

- YES. Without a doubt, the extra help and practice from labs helped my grade in COP2271. I had access to the professor and TA's outside of office hours, and the labs were always just as or more difficult than the homework and exam problems, so I always felt prepared. I highly appreciate the effort and thought put into making fun/interesting lab topics.

**Discussion**

Students mostly either strongly agreed or somewhat agreed in a positive manner to each assessment question. Approximately 72% students strongly agreed that the labs were interesting indicating that the chosen programming challenges resonated across the different engineering disciplines and gender. Also, 85% of students would strongly recommend the lab to their peers showing students believe the lab to be useful in learning programming instead of simply a required component to the class. It is interesting to see that students did not all agree that lab was a motivation to regularly keep pace with lecture. This is perhaps a downside of the nature of group work; students could depend on peers to answer questions about material shown in lecture. Lastly, 72% of students were able to make strong connections to the activities and how their particular engineering discipline fits into society. It is important to keep making strides here so students see the relevance of their efforts.

It is difficult to compare these results to the previous curriculum design since a comparable course assessment is not available. The traditional university-wide course assessments were completed but the response rates were low for the laboratory sections, making it difficult to make any useful comparisons.

The assessments show promising results, but there are still key changes that can help students with different learning styles. For example, students often request additional practice problems that are shorter and allow students to repeatedly practice a concept. This is in direct contrast to the new laboratory style of having students work in groups to solve a few select, challenging problems. Ideally the laboratory can provide a bank of problems with solutions and explanations that students work on at their own pace for further practice. In addition, debugging practices are often left to the student to discover but should be explicitly taught due to the relatively complexity of the new problems. Learning these tools will help students diagnose and fix their own coding mistakes.

**Conclusions**

This paper presents an updated curriculum centered on problem-based learning and just-in-time teaching for a weekly, one-credit programming laboratory course. The class must support a wide variety of engineering majors while reinforcing programming fundamentals from lecture.

The labs were designed to peak student interest while forcing students to practice current concepts with questions that challenged their understanding. Assessments and student feedback show the changes are largely successful, with students strongly recommending the course to their peers. Future work includes improving the assessment questions to track student progress along each learning objective, specifically showing the improvement of students without prior programming experience. Furthermore, students will be tracked through future engineering classes to quantify if the laboratory successfully prepares them to apply programming skills to new subjects.

## References


1. Felder, Richard M., and Rebecca Brent. *Teaching and learning STEM: A practical guide*. John Wiley & Sons, 2016.
2. Prince, Michael J., and Richard M. Felder. "Inductive teaching and learning methods: Definitions, comparisons, and research bases." *Journal of engineering education* 95.2 (2006): 123-138.
3. Albanese, Mark A., and Susan Mitchell. "Problem-based learning: a review of literature on its outcomes and implementation issues." *Academic medicine* 68.1 (1993): 52-81.
4. Hmelo-Silver, Cindy E. "Problem-based learning: What and how do students learn?" *Educational psychology review* 16.3 (2004): 235-266.

## Appendix A: MATLAB Paint Clone Skeleton

```
function [] = PaintSkeleton(width,height)
    % ---------------------------------------------------------------------
    % Create a blank window to display
    % ---------------------------------------------------------------------
    window = figure('Visible','off','Position',[0,0,width,height]);
    movegui(window,'center');

    % ---------------------------------------------------------------------
    % Add a drawing space to the window
    % ---------------------------------------------------------------------
    graphics = axes('Units','Pixels','Position',[0,0,width,height]);
    axis([0 width 0 height])
    set(graphics,'Ytick',[],'Xtick',[]);

    % ---------------------------------------------------------------------
    % Use the set function to update different properties of the window
    % ---------------------------------------------------------------------
    set(window,'MenuBar','none','ToolBar','none');
    set(window,'KeyPressFcn',@KeyPress);
    set(window,'WindowButtonDownFcn',@MouseDown,'WindowButtonUpFcn',@MouseUp);
    set(window,'WindowButtonMotionFcn',@MouseMove);
    set(window,'Name','Paint Clone','Visible','on');

    % ---------------------------------------------------------------------
    % KeyPress function that runs anytime the user presses a key
    % ---------------------------------------------------------------------
    function [] = KeyPress(~,event)
        keyDown = event.Character;
    end

    % ---------------------------------------------------------------------
    % MouseMove function that runs anytime the user moves the mouse
    % Inputs: ~, ~
    % Outputs: None
    % ---------------------------------------------------------------------
    function [] = MouseMove(~,~)

    end

    % ---------------------------------------------------------------------
    % MouseDown function that runs anytime the user presses mouse button
    % ---------------------------------------------------------------------
    function [] = MouseDown(~,~)

    end

    % ---------------------------------------------------------------------
    % MouseUP function that runs anytime the user lets go of the mouse button
    % ---------------------------------------------------------------------
    function [] = MouseUp(~,~)

    end
end
```

## Appendix B: C++ Paint Clone Skeleton

```
// Headers and namespaces
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Network.hpp>
#include <SFML/OpenGL.hpp>
using namespace sf;

int main()
{
    //Create a window that we can draw in
    RenderWindow window(VideoMode(800,800),"Paint Clone");

    //Texture and sprite
    Texture canvas;
    canvas.create(800,800);
    Sprite drawCanvas(canvas);


    // First required while loop - runs until the window is closed!
    while ( window.isOpen() ) {
        // Second required while loop for handling events
        Event event;
        while ( window.pollEvent(event) ) {
            if (event.type == Event::Closed)
                window.close();
        }

        //Check if mouse moved
        Vector2i mousePos = Mouse::getPosition(window);

        //Check various mouse buttons
        bool rightButton = Mouse::isButtonPressed(Mouse::Right);
        bool middleButotn = Mouse::isButtonPressed(Mouse::Middle);
        bool leftButton = Mouse::isButtonPressed(Mouse::Left);

        //Check if a particular key has been pressed
        if ( Keyboard::isKeyPressed(Keyboard::Escape) ) {
        }

        //Draw in the window
        window.clear(Color::White);
        window.draw(drawCanvas);
        //Add additional drawing routines here

        canvas.update(window);
        window.display();
    }
    return 0;
}
```