An Interactive Programming Course Model for Mechanical Engineering Students

Prof. Shanon Marie Reckinger, Fairfield University

Shanon Reckinger joined the department of Mechanical Engineering at Fairfield University in Fall 2011. She received her PhD in Mechanical Engineering at the University of Colorado Boulder in August of 2011. Her research interests include ocean modeling, computational fluid dynamics, fluid dynamics, and numerical methods. At Fairfield she has taught courses in thermodynamics, numerical methods (graduate), fluid dynamics, gas dynamics (graduate), computational fluid dynamics (graduate), fundamentals of engineering, mathematical analysis in MATLAB.

Dr. Scott James Reckinger, Brown University

Scott Reckinger is a postdoctoral research assistant in the department of Geological Sciences at Brown University. Scott received his PhD in Mechanical Engineering in May, 2013 at the University of Colorado Boulder (UCB). His research interests include climate modeling, computational fluid dynamics, and numerical methods. He has taught and guest lectured in fluid dynamics, numerical methods, classical physics, mathematics for engineers, and computational fluid dynamics at Fairfield University and UCB.

An Interactive Programming Course Model for Mechanical Engineering Students

Abstract

Programming is a crucial skill for today's engineering student. The majority of mechanical engineering programs in the US include an "introduction to programming" course taken during the first or second year. The primary goal of the course is to providing students with the basic programming techniques that are required to excel in specific mechanical engineering fields of study. Additionally, the course aims to develop a variety of skills that transcend all scientific disciplines, including problem solving, logical reasoning, debugging, and software training. A course in programming can be challenging for many students choosing to major in mechanical engineering. The major attracts students with diverse backgrounds and a wide variety of academic interests. It is uncommon for students to choose to study mechanical engineering because of their interest in programming or modeling. This often leads to a disconnect between the students and the instructor, which can create an intimidating classroom environment. The work presented here is driven by these findings.

A new programming course has been developed to address the problems existing in the original course model, which include: (a) the course being offered outside of an engineering department, (b) the extreme variability in the rate at which the students comprehend the material, and (c) the frustration of new programmers, especially with debugging. Backward course design[1] was used to redesign the course, addressing all of the existing problems. First, the new course focuses on engineering specific computational applications, is taught by a Mechanical Engineering professor, and uses a more practical programming language, MATLAB. Thus, the essentials of programming are introduced within a focused framework that cultivates the development of analytical tools commonly used in engineering disciplines, such as statistics, data analysis, numerical differentiation and integration, and Fourier analysis. Second, the Process-Oriented Guided Inquiry Learning (POGIL) method[2] is used so that students are self-guided through part of the instruction. Lastly, class time is organized in such a way that the instructor spends over half of the time working directly with individuals and small groups. This gives the students an opportunity to have explanations individually catered to their level of understanding, as well as plenty of time for peer and instructor assistance with debugging.

The course initially ran under the new model in Spring, 2013. The course ran for 15 weeks and had 37 students split into two different sections. There were no teaching assistants. Feedback from the students indicated that they benefitted greatly from the course design. Improvements for the second iteration of the new course model, which will occur in Spring, 2014, include lengthening the course from 2.5 hours per week to 4 hours per week, utilizing more traditional lecture, incorporating class discussions, adding student created supplementary video content in the essence of classroom flipping[3], and integrating an overarching humanitarian theme to all assignments in an effort to support the liberal arts goals of the university.

Introduction/Motivation

This course design was motivated by many problems with the existing programming course, but focused on addressing three issues:

(a) the course being offered outside of an engineering department,
(b) the extreme variability in the rate at which the students comprehend the material, and
(c) the frustration of new programmers, especially with debugging.

Issue (a) was straight-forward, instead of the course being taught by computer science faculty, a mechanical engineering faculty developed and taught the course. While straight-forward to solve, it is an important point to drive home. Computer science departments program for different applications than mechanical engineers do. The majority of mechanical engineers will not do a substantial amount of low level programming in their careers. However, it is becoming very common for mechanical engineers to incorporate high level, simple programming techniques in their day-to-day work. This could be for data analysis, programming manufacturing equipment, modeling for mechanical design, control theory and robotics, etc. There are several advantages to having the course taught within the department.

The first advantage is getting the freshmen into an engineering course as early as possible by having a mechanical engineering faculty teach the course. Engineering programs require many prerequisites in mathematics, physics, chemistry, computer science, and more. Thus, students don't start taking substantial engineering courses from engineering faculty until their sophomore year. Although the foundational courses in the physical sciences are definitely crucial aspects of any engineering program, it is hard to argue that limited interactions with engineering faculty would be beneficial for the program's retention. Since a programming course can be offered during the first year, it is a good opportunity for mechanical engineering programs to teach the course within the department to give the students another opportunity to explore the engineering field more directly.

The second advantage is the freedom of choice of the programming language. Computer science departments are not always familiar with the programming languages most prevalent in the engineering disciplines. It has become a common problem in engineering programs that students learn to program their freshmen year in a language chosen by the computer science department, and then later in the academic program when the students are asked to apply that knowledge to a new language or new problem, they are unable to show even a basic level of proficiency. For example, the Electrical Engineering program at University of Texas, El Paso introduced a programming course taught by EE professors because they found that only 20% of their EE seniors were proficient in programming[4]. Initial results show that it was effective. Not only does an engineering professor teach it, but it also combines programming and mathematics. This is similar to the course being presented in this paper. Other people have found that MATLAB is an important language to use when teaching programming because of its simplicity. Even in courses that teach C-programming, MATLAB is used to target specific concepts.[5] Another study showed that when teaching C programming, that the students have a problem understanding the concept of arrays, dealing with the syntax of the language, designing the organization of the program, and understanding the concept of flow control such as looping and branching or

function calls. To mitigate this, they used MATLAB to focus only on loops and Excel to teach arrays.[6] There are many higher division classes within mechanical engineering and other engineering disciplines that are using MATLAB to solve problems and understand concepts.[7] Therefore, it is becoming more common in mechanical engineering programs to simply teach MATLAB in the introductory programming class.

The third advantage is freedom to design the course, which employs programming techniques to applications pertaining to the field of mechanical engineering (or engineering, in general). For example, computer science departments will often create programming assignments which have the student create a simple game or puzzle. Or they will write a script that creates a website or a software tool. While those are great (and even fun) assignments, they are not applicable to most mechanical engineering students career interests. Therefore, teaching the course within the home department allows full control of these applications. The students can program assignments relating to data analysis, statistics, numerical methods, and other mathematical techniques almost all engineers use on a regular basis. Many of them are relevant to the courses the students are enrolled in concurrently (such as Calculus and Physics).

Issue (b) is a common complaint among instructors of introductory programming courses, which is that the students background is too diverse. This leaves the instructor with a challenging classroom to teach in. Programming is not a standardized course in high schools. Some students might have already taken a programming course in high school, while other student might not even know what programming is. There are several published theories on how to teach programming[8 9], however, one interesting study pre-tested the student prior to taking an introductory programming class.[10] The pretest was comprised of general logic and math questions that were predictive of programming ability. They found that from over 800 student, their performance on the pretest correlated with their success in the programming course. This is an interesting concept relating to students' programming background. Perhaps previous programming experience is not as influential in the success of learning how to program, as is basic mathematical and logical skills. This is evidence that teaching students to program is less about their programming backgrounds and more about the backgrounds in math and logic. Therefore, effective teaching methods would focus more on those types of concepts. There have been several creative approaches to teaching programming such as using inductive learning and robots, which was done at West Virginia University.[11] Inductive learning is when students are given a problem to solve first. They try to solve that problem using what they know, which often is not enough and work to acquire that knowledge in order to solve the problem. Inductive learning is a much more natural way of learning and it the normal way most people go about learning otherwise. In your daily lives, you likely did not learn to use a knife, grate cheese, and operate the oven before understanding the purpose of those skills. The natural process is that you get hungry, you decide you want to eat pizza, you decide to make pizza yourself, you look up a recipe on how to cook pizza, you discover to need to know to chop, grate, and turn on an oven, so you learn to do those things. It is fairly natural to desire to obtain that knowledge when you have a purpose for it. Inductive learning fits well with the POGIL method. Instead of teaching students exactly the syntax and functionality of a particular feature of MATLAB, you just ask them to show you via a POGIL worksheet. If you want students to know how save an array of data in the MATLAB workspace, just ask them to do it and they will figure it out. Others have found that it is important to take a modular approach when teaching programming.[12]

That is, breaking a bigger computer program into smaller tasks when teaching freshmen and sophomores at an introductory level. The course presented here incorporated aspects of both of these approaches. Instead of robots, other interesting, fun, motivating assignments were designed. Lecture was often spent carefully breaking down logic into more manageable pieces.

Issue (c) is possibly one of the most challenging aspects of teaching introductory level programming. Even in a user-friendly program like MATLAB, student are often overwhelmed and frustrated with error messages and debugging their code. Even when every effort is taken to explain how, lecture on, quiz on, and assess students ability finding and fixing a bug, the only tried and true effective method is for students be forced to do it repeatedly. They need to write code, run code, see that it does not work, read the error message, and figure out what is wrong.

Of course, they need a lot of help when they first start doing this. This was the motivation in the overarching design of the course. The challenge is how to maximize the amount of time students have access to help with debugging. For this course, the POGIL method was used so that students could spend more time in class working with MATLAB "hands on" and less time following a lecture, watching a demo, or trying to follow along with a demo. That way, the instructor spent 30-45 minutes twice a week with each section working with students one-on-one or in small groups. While the instructor was answering a question or check progress on an assignment for one student or a small group of students, the rest of the class was working through a POGIL worksheet, on a programming assignments, or preparing their toolboxes for the exam. Incorporating a lab-like setting into the class time also allowed for students to work together to solve the problems and debug each other's codes.

Course Design

Backward course design[13] was used to build the course from a blank slate. Starting with the end goal in mind, three course goals were chosen. These three goals are the backbone of the course. They answer the question, "what would you like your students to take away from the class?"

Next, seven measurable course outcomes were laid out and all linked back to at least one of the course goals. These course goals and outcomes can be found in Figure 1. Each of the course outcomes was then also linked to the Accreditation Board for Engineering and Technology (ABET)[14] student outcomes and Bloom's taxonomy's cognitive level[15]. Since this is an introductory course taught to engineering freshmen, the highest cognition level expected is "application".

Course Goals

I. Develop a foundational understanding of computer programming and how it is applied in the field of engineering.
II. Develop an understanding of mathematics, numerical methods, and statistics especially relevant to the field of engineering.
III. Encourage methodical, orderly, and disciplined study of engineering.

Course Outcomes

1. Demonstrate introductory level computer programming skills including differentiating between data types, array

creation and manipulation, the use of control flow, defining functions, and read and writing data.. [I] (a, e, g) {application}
2. Show proficiency in MATLAB including the understanding of the workspace and GUI, using m-files, graphics and plotting, and vector storage. [I] (k) {knowledge}
3. Demonstrate mastery of mathematical, numerical, and statistical engineering topics such as matrix algebra, data analysis and statistics, data interpolation, curve fitting, Fourier analysis, integration, differentiation, and optimization. [II] (a, e, g) {application}
4. Organize a concise MATLAB binder summarizing all topics relevant to the course outcomes. [III] (g, i) {knowledge}
5. Employ the ability to learn independently or to know when to ask for help, to most efficiently and successfully acquire knowledge. [III] (d, g) {application}
6. Comprehend the ethics of programming. [I] (f) {comprehension}
7. Identify how programming and mathematical content applies to the field of engineering. [I,II] (h) {knowledge}

[ ] course outcome link to course goal
( ) lower case letters (a-k) link to ABET student outcomes
{ } Bloom's taxonomy's cognitive level of learning (knowledge, comprehension, application, analysis, synthesis, evaluation)

Figure 1 – An excerpt from the syllabus showing the course goals and outcomes mapped to each other, ABET student outcomes, and Bloom's taxonomy cognitive level.

Once the course outcomes were determined, the course assessment methods were designed in order to assess whether or not students were achieving those outcomes. This will be discussed in detail in the following section, but the assessment methods were class participation (10%), MATLAB binder (40%), midterm exam (20%), final exam (30%). Additionally, the instructional strategies were designed in order to facilitate student learning with respect to the pre-determined course outcomes. These strategies included POGIL worksheets, which are student guided learning, short and concise traditional board lecture, MATLAB demos, daily clicker quizzes, programming assignments, a mid-semester survey (with a report of results and discussion), and in-class debugging (peer and instructor). Both the assessment methods and instructional strategies will be discussed in detail in the following section.

Pedagogical Approach

The course was designed to utilize several newer pedagogical approaches that have become more popular in engineering education. These include:

- daily course polling/quizzing through the TurningPoint Clicker System,
- student guided instruction through the use of instructor created worksheets based off of the POGIL method,
- active learning through limited lecture and limited instructor demoing.

However, the course also incorporated tried and true traditional engineering education pedagogical approaches, including:

- concise board lectures highlighting difficult concepts, organizing content, or clarifying more complex ideas,
- well thought out, short, to the point MATLAB demos introducing new features that the students will be exploring that week,

- traditional, written tests with no access to computers or other technology,
- required student created crib sheets to use on exams, one per topic.

These pedagogical approaches, both new and old, will be explained in detail in the following paragraphs. This will be presented through a detailed explanation of the instructional strategies and assessment methods.

Instructional Strategies:

The overview of the course schedule can be found in Table 1. The course was broken down into 13 topics. Depending on the complexity of the topic, one or two class sessions were used to deliver that topic. Each topic corresponded with a chapter in the book and had its own assignments, which will be discussed later.

Session	Topics Covered	Reading	Assignments Due
1	T1: Intro to Computers	Mentor	
2	T2: MATLAB Basics	Ch. 1-4	WS1, TB1
3	T3: Numeric Data Types, Character Strings	Ch. 7, 9	WS2, TB2
4	T4: Arrays, Array Operations, Multidimensional Arrays	Ch. 5, 6, 8	WS3, TB3
5	T5: Relational and Logical Operations, Control Flow (Loops)	Ch. 10, 11	WS4, TB4
6	T5: Relational and Logical Operations, Control Flow (Loops)		
7	T6: Functions	Ch. 12	WS5, TB5, PA2
	No Class. President's Day.		
8	T6: Functions		PA3, PA4
9	T7: File and Directory Management	Ch. 13	WS6, TB6
10	T8: Graphics and Plotting	Ch. 25-30	WS7, TB7, PA5
11	T8: Graphics and Plotting		PA6
Midterm	Midterm Exam, Topics: T1-T8		
12	T8: Graphics and Plotting & Return/Review Exams		
	No Class-Spring Break		
13	T9: Data Analysis and Statistics	Ch. 17	WS8, TB8
14	T9: Data Analysis and Statistics		PA8
15	T10: Curve Fitting	Ch. 19	WS9, TB9
16	T10: Curve Fitting		PA9
	No Class. Easter Break.		
	No Class. Work Day.		
17	T11: Data Interpolation	Ch. 18	WS10, TB10
18	T11: Data Interpolation		PA10
19	T12: Integration and Differentiation	Ch. 23	WS11, TB11
20	T12: Integration and Differentiation		PA11
21	T13: Fourier Analysis	Ch. 21	WS12, TB12
22	T13: Fourier Analysis		PA12
23	Work Day		WS13, TB13
24	Review for Final Exam *(optional)*		PA13
Final	Final Exam time: 3-6 pm, DSB 104A (Dining Hall) NOTE: **Both** sections will take the exam at this time.		

Table 1 – Overview of the course schedule.

Each session followed the same overall schedule, which is shown in Table 2. The class starts promptly (within a minute) of the posted start time in the syllabus. While this seems like a nit picky detail, it is a vital component of developing good rapport with the students and also

contributing to the success of course goal III, which is "encourage methodical, orderly, and disciplined study of engineering". To motivate the importance of being in class on time and ready to learn, a clicker quiz is administered immediately. Multiple choice or true/false questions are asked based off of the topics covered in the previous class. This clicker poll serves many purposes: attendance, review, an additional chance to recall material, and instant feedback for the instructor on how well the class is grasping concepts. Figure 2 shows an example of two different types of clicker questions, one based on theory and one based on programming. Two to five questions were given per day, with a total of 66 throughout the semester. Students participation in the questions were incorporated into their participation grade. Although their performance on the questions were not incorporated into the grade, awards were given out at the end of semester relating to clicker scores and participation. The high percent correct was 77%, the lowest was 31%, and the average was 54%. These results are reasonable considering the material was all only one session old.

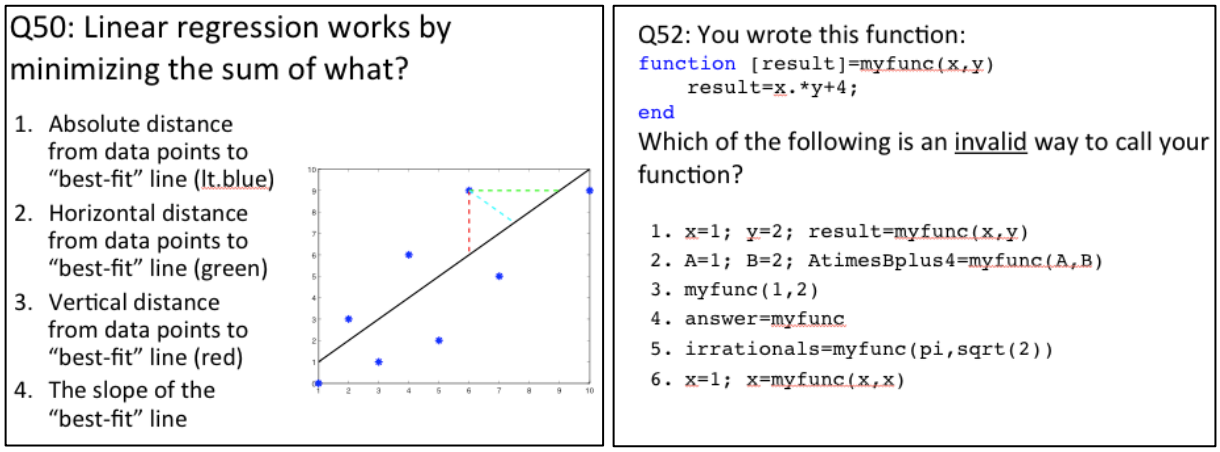

Figure 2 - Two examples of clicker questions used during daily clicker quizzes.

Activity	Time
Clicker Poll/Quiz	3-5 min.
Announcements	2-5 min.
Board Lecture	10-15 min.
MATLAB Demo	5-15 min.
Student Work Time	30-45 min
Total	**75 min.**

Table 2 – Daily schedule for the class.

Following the clicker quiz and announcements, a concise board lecture took place based on the topic of the day. The lecture was used to present difficult concepts, to organize material, or to clarify specific strategies needed for the assignments they were working on. Everything that was presented was needed for one or more of the assignments they were working on. Topics during the first half of the semester were covering how to use the software and basic programming techniques. An example of board would be to explain the difference between a do loop and a while loop. Or, the board lecture might be writing a pseudo-code and going through the step by step logic of how a computer could solve one of their programming assignments.

Either following or interlaced into the board lecture, a MATLAB demo would be presented to show how to apply that concept to the software. At the beginning of the semester, this would be showing how to define variables, arrays, and MATLAB syntax in the Workspace. Some students followed along on their own computers while others just took notes. Later in the semester, these demos would program smaller pieces or modules of the programming assignments that they are working on. For example, if they needed to program a second derivative using a central differencing scheme, the MATLAB demo might show them how to do a first derivative using a backward differencing scheme. Or the demo might show them how to write a function and then call that function from another function, which they will need for their assignment.

Finally, the last 30-45 minutes of class is dedicated to student work time. This is when students can pick up graded assignments, check the instructor's solution manual (not available online), ask questions on content (from peers and instructor), get help debugging (from peers and instructor), and get assignments "signed off" by the instructor. This very busy time almost always bled into the next session. During this time, the instructor was able to meet with students one and one and by the end of the semester, was very familiar with the programming skills of each and every student from that experience alone.

Assessment Methods

As mentioned before, assessment methods were class participation (10%), MATLAB binder (40%), midterm exam (20%), final exam (30%).

Class participation was evaluated based off of attendance, clicker question participation (not score), and observations of work ethic during class. If students were browsing the web, off task, repeatedly forgetting materials or laptop, etc. their attendance grade was affected. Out of all 37 students, the attendance rate was 99%, only 11 absences total.

The MATLAB binder was the most heavily weighted assessment method. For each of the 13 course topics, the students were required to complete a Toolbox, a Worksheet, and a Programming Assignment. The toolbox was a single sheet of notes (a crib sheet) that they completed after finishing the worksheet and programming assignment. This was graded and to be used on the exams. An example of a student's toolbox is shown in Figure 3.

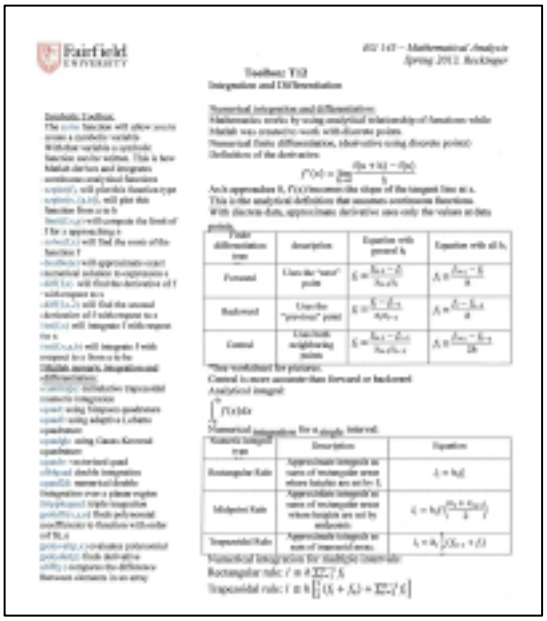

Figure 3 – An example of a student's toolbox from Topic 12.

The worksheet was designed by the instructor so that the students would self guide themselves through the material to supplement the other instructional strategies. Figure 4 shows an example of a few questions on a worksheet for Topic 4, on arrays. An important point to note is that the material in the worksheets was not necessarily covered during the board lectures or MATLAB demos. Therefore, the students needed to use their resources during the class to find the answers. Although for basic programming and MATLAB training this is not exactly inductive learning, it is in a similar spirit. The students want to figure it out because it allows them to complete the worksheet. The worksheets are turned in a graded as part of the MATLAB binder grade.

1. What is an array?
2. **Creating a simple array:** In the MATLAB Command Window, create an array using the variable name `angles` and fill it with 10 angle values (use: 0°, 30°, 60°, 90°, 120°, 150°, 180°, 210°, 240°, 270°). Copy down the command to do this here. *(NOTE: It is not necessary in any of these exercises to copy down the output or what is printed when the command is executed. Simply copy down the command.)*
3. **Operating on a simple array**: Perform two operations on this array. First, convert all the angle values from degrees to radians. Second, find the cosine of all the angles using the original created array called `angles`. Copy down each command here (each should be a single line and each command should operate on the entire array at once).
4. **Array Addressing or Indexing**: Show how to access a single array element of an array (do this by accessing and displaying in the command window the value of 120° from the array `angles`). Copy down the command here.

Figure 4 - A snapshot of a worksheet on arrays to provide an example of a POGIL guided exercise.

Students apply their programming and logic skills to a programming assignment for each of the topics. Programming assignments are more complex problems that must be solved by writing an m-file and running the code. Figure 5 and 6 show an example of a programming assignment and the solution. The assignments are designed to utilize control flow, arrays, function calls, and all of the other programming techniques covered in the first half of the semester. The last half of the semester was dedicated to applying these techniques to mathematical and statistical tools that engineers use on a regular basis. This included data analysis, statistics, curve fitting, interpolation, numerical differentiation, numerical integration, and Fourier analysis. Unlike the tool boxes and worksheets, programming assignments were not turned in for grading. Instead, during work time in class, students must get their programming assignments "signed off" by the instructor. The instructor would have the student run the program, ask a few questions, and give them a binary score on the completion of the assignment. This was more efficient than submitting programs and running each individually for evaluation. This process also allowed the instructor to do an unofficial oral exam to evaluate the students learning. The one-on-one interaction during sign offs provided extremely value predictive data on students learning and their potential success on other course assessment.

PA12– Integrate and Differentiate Data from a stage of the Tour de France
Function purpose: Write a function that integrates and differentiates data from the 2012 Tour de France rider Chris Anker Sorensen (Stage 16).

1. Create a new .m file function. Give it a function name and a file name: `pa12`. It will have no inputs and no outputs.
2. Load in all the data from tourdeFrance.mat. Type `whos` underneath the `load` command to see what data was loaded in. Units are as follows: `time` [hr], `speed` [km/hr], and `elevation` [m].
   NOTE: This is the speed and elevation of racer Sorensen from Stage 16 of the 2012 Tour de France. The stage was a total 6 hours!
3. In the first of two subplots, using a `plotyy,` plot time vs. speed on the left axis and time vs. elevation on the right axis. Label the each axis and title the plot (include units).
4. Approximate the total length of Stage 16 in km (i.e. the total distance traveled) using the trapezoidal rule (i.e. integrate the speed of the race over the 6 hours). Write your own trapezoidal rule, do not use MATLAB intrinsic functions. This can be done in a single line of code or by using a `for` loop. Display total distance traveled (include units) and check that your answer is right (you can check here: http://www.letour.com/le-tour/2012/us/overall-route.html)
5. Approximate the derivative of elevation over time using a forward difference approximation. Write your own difference approximation, do not use MATLAB intrinsic functions. This is the vertical speed of the racer.
6. In the second of two subplots, plot time vs. the vertical speed of the racer calculated above. Label each axis and title the plot (include units). Verify that your derivative calculation makes sense.

Figure 5 – An example of a programming assignment description.

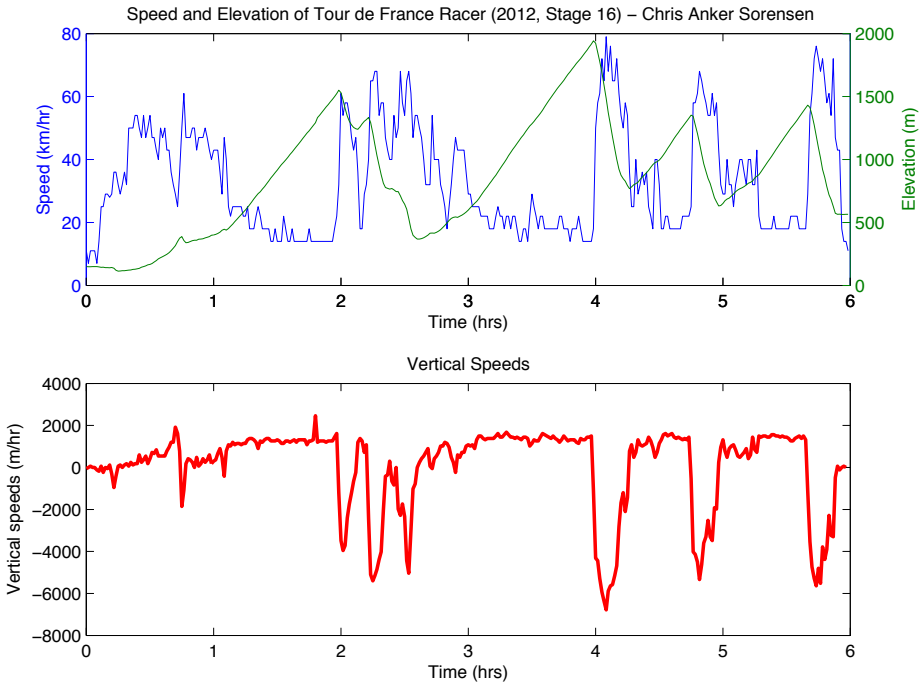

Figure 6 – An example of a solution to a programming assignment (PA12).

Finally, 50% of the student's grade was based off of their performance on the two exams: a midterm and a final. The exams were closed book and close electronic devices. However, they were able to use their MATLAB binder during the exams. Exam questions mainly comprised of testing the mathematical theory or testing their programming ability. For example, they had to hand write MATLAB code using proper MATLAB syntax. An example of this is shown in Figure 7.

1. *(20 pts)* Given the following data:

time (hrs)	0	1	2	3	4
cost ($)	0	100	150	175	200

Write code that does the following:

i. Write the function declaration line. Give it the function name: **`calctot`**. The function will have one input (**`const`**) and one output (**`tot`**).
ii. Stores the data above in two arrays: **`t`** and **`c`**
iii. Initialize an array (give it variable name: **`tot`**) that is the same size as **`t`** and fill it with zeros (do not use any numbers for this line).
iv. Computes the total cost using the following formula (where N is the size of **`t`** or **`c`** and p is the input **`const`** that is read in by the function) and store in **`tot`**:

$$\text{Total}=\sum_{n=1}^{N}(pt_n + c_n)$$

Figure 7 – An example exam question, which assesses students programming proficiency.

Survey Results and Feedback

There were two forms of evaluation used to study the effectiveness of this new course design. The first was mid-semester survey administered by the instructor. The second was the course evaluations administered by the Individual Development and Educational Assessment (IDEA) Center[16] at the end of the semester.

Mid-semester Survey

The mid-semester survey was a series of 8 questions and was administered via a non-anonymous clicker question in class questionnaire. The questions and answer options are listed in Table 3.

On a scale of 1-9, how much are you enjoying this class (1-not at all, 9-the most)	1	2	3	4	5	6	7	8	9
What is your opinion about lecture time?	Want more			Like current amount			Want less		
What is your opinion on work time during class?	Want more			Like current amount			Want less		
What materials did you find most useful on the exam?	WS	TB	PA	All	None	WS & TB	TB & PA	WS & PA	
Did you find that the material on the exam matched the material we focused on in class (lecture and all assignments)?	Yes				No				
How did you feel you performed on the exam?	A		B		C		D		F
How much of the worksheets do you do unassisted?	Always work alone.		Always work with others		Mostly work alone		Mostly work with others		It depends
How much of the programming assignments do you do unassisted?	Always work alone.		Always work with others		Mostly work alone		Mostly work with others		It depends

Table 3 – Mid-semester survey conducted by instructor via in class (not anonymous) clicker questions.

One of the motivations of conducting the survey was to see student's perception of the pedagogical approach. A common complaint from students who are taking a course which uses the POGIL approach is something along the lines, "The instructor doesn't teach the material, we are basically teaching ourselves." Therefore, the students were surveyed about how they felt about the way class time was spent. Figure 8 shows to pie charts summarizing the results. It was found that 56% of the students wanted more lecture time, 44% thought there was the right amount, and none of the students wanted less lecture time. Therefore, starting at mid-semester, more lecture time was added. Since most classes are still operating under the traditional lecture based class structure, many students are not yet comfortable with little or no lecture. When asked about work time during class, 68% thought there was the right amount and rest were split between wanting more or less.

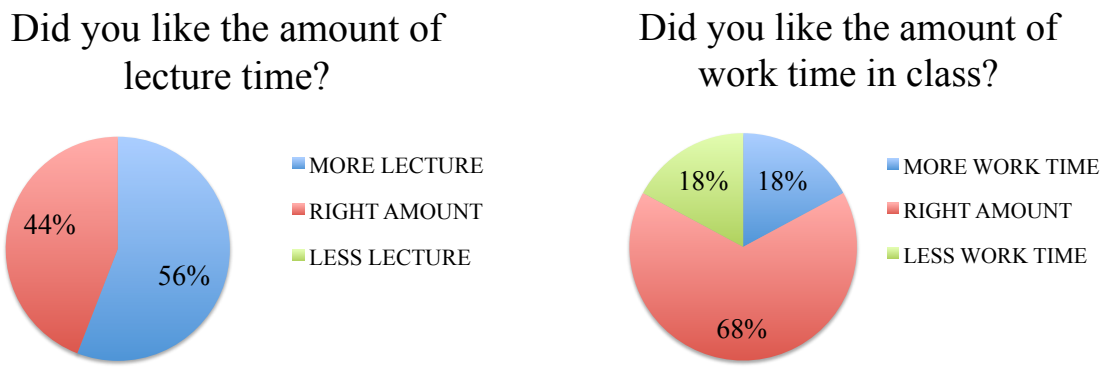

Figure 8 – Feedback from students on the amount of time on lecture vs. amount of in class work time.

For fun, students were asked how much they were enjoying the class. It was a fairly subjective question and results are likely not extremely accurate since the survey was not done anonymously (even if it was, the students would might not trust that it was actually anonymous). Therefore, students were generally positive about the class, as shown in Figure 9.

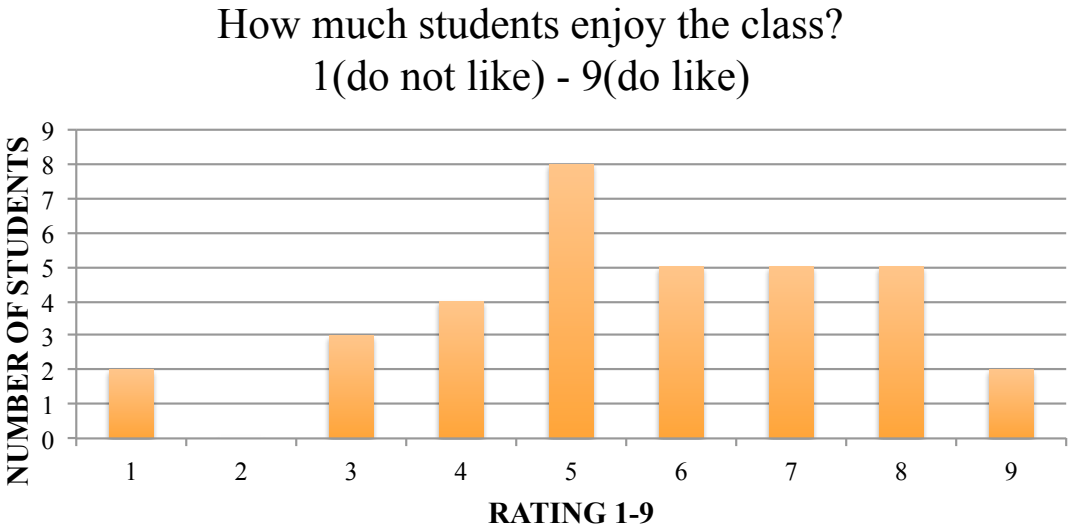

Figure 9 – Feedback from students on how much they like the course.

The survey was done in the class session immediately following the midterm exam and was conducted prior to handing back the graded exam. Students were surveyed to see how they felt they did on the exam, giving themselves an average grade of an A, B, C, D, or F. There predicted scores were conservative and followed a traditional Gaussian distribution. Their actual grades were inflated quite a bit due to the curve applied, with most students falling in the B range. Figure 10 shows details of these distributions.

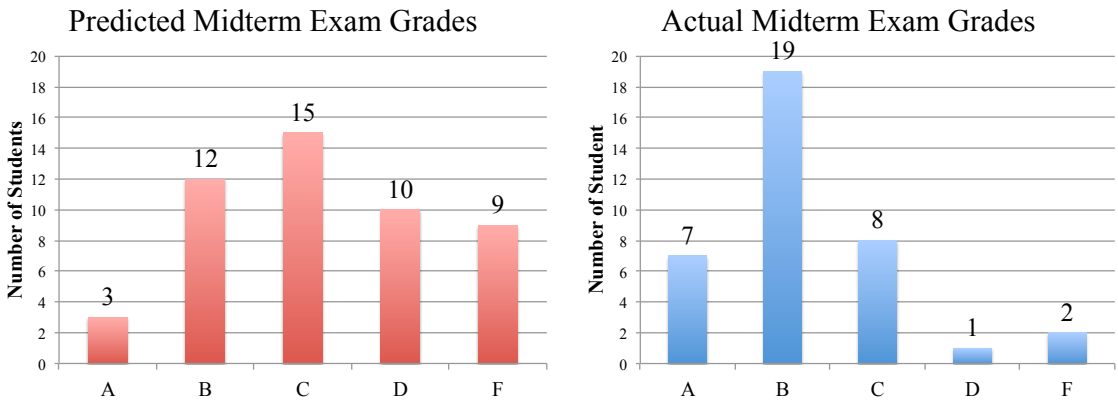

Figure 10 – A comparison of how students predicted they did on the midterm exam versus how they actually did.

Finally, some correlations were done to study the effectiveness of different pedagogic strategies. Students were surveyed to see how much they work alone and how much they work with others, see Table 3 for exact questions and answer options. Their answers were correlated with their performance on the midterm exam (Figure 11) in hopes of providing them with proper feedback on how to best improve their learning in the class. Results showed that students who reported "working mostly with other" on worksheets had the highest average score on the midterm exam. Students who answered "it depends", had the lowest average score. This indicates that working in groups on the worksheets but also doing some of the work individually proved to be the most effective way to learn the material. On the other hand, there did not appear to be any correlation with working with others on the programming assignments and performance on the midterm exam.

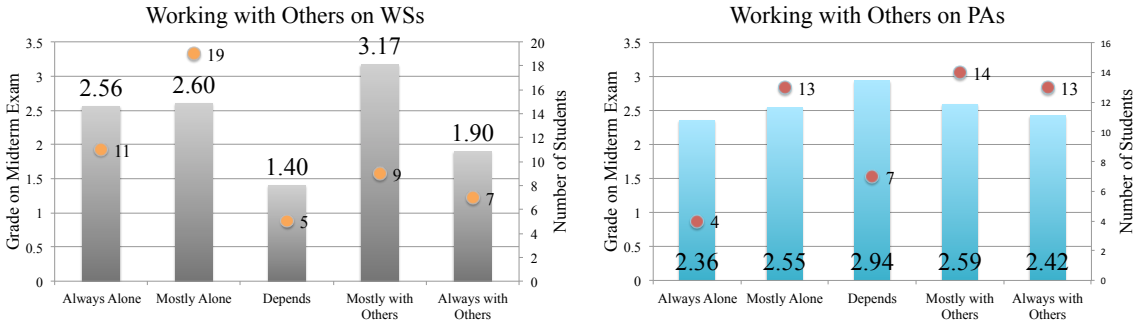

Figure 11 – A graph showing the correlation between students working with their peers on the worksheets or programming assignments and their grade on the midterm exam.

An additional correlation was investigated between working with others and actual scores on the worksheets. Figure 12 shows that there is not much of a correlation. This is shows that even though students are getting the correct answers on worksheets regardless of whether they work along or with others, this is not indicative of whether or not they are truly learning the material (which is what the exam can demonstrate).

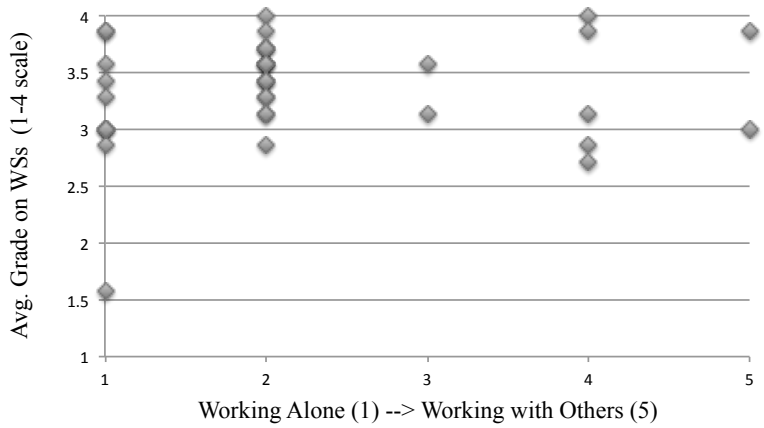

Figure 12 – A graph correlating the average grade on POGIL worksheets with how much students reported they worked with their peers on the worksheets.

The students were also asked if they felt that the test material was sufficiently covered in the different instructional strategies and assessment methods. Figure 13 shows that 76% of students felt that the test matched. Also, as expected, the average score on the exam (using a 4.0 scale) was much higher for those who answered "yes" vs. those who answered "no".

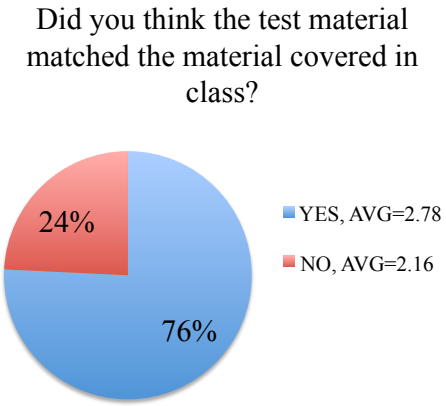

Figure 13 – Pie chart summarizing student responses to how the test material corresponding to what was focused on in class. Average score (using a 4 point scale) is shown for each group of students.

Lastly, the students were surveyed to see if they felt the three assessment methods (tool boxes, worksheets, and programming assignments) were useful on the exam. As seen in Figure 14, overall students used all three of the methods. The toolboxes were used to least, which is probably because they are the most open-ended of the three.

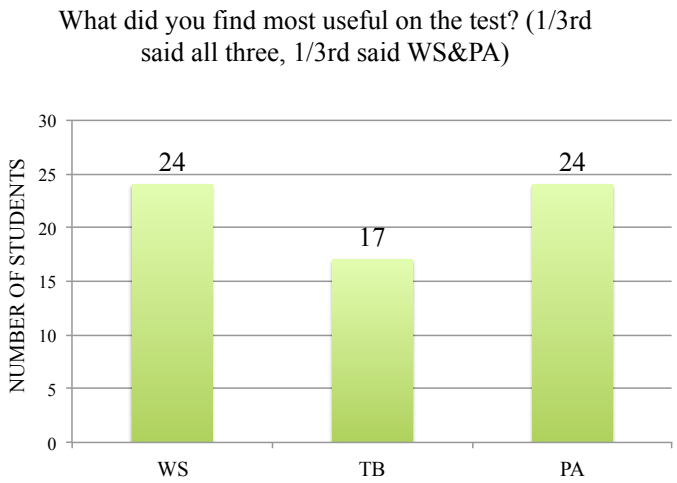

Figure 14 – A bar graph summarizing the results of what assessment method students found to be most useful on the midterm exam.

IDEA Student Evaluation Results

As part of the IDEA student evaluation process, the instructor chooses 3-5 learning objectives and indicates them to be either "important" or "essential" (i.e. more important). For this course, the instructor chose two "essential" learning objectives:

21. Gaining factual knowledge (terminology, classifications, methods, trends)
22. Learning fundamental principles, generalizations, or theories.

And then chose two "important" learning objectives:

23. Learning to apply course material (to improve thinking, problem solving, and decisions)
29. Learning how to find and use resources for answering questions and solving problems.

The students were surveyed to rate their progress on these objectives using a 1-5 scale. The scale was 1-no, 2-slight, 3-moderate, 4-substantial, 5-exceptional progress. The course was comprised of a total 37 students split into two sections a morning section (Sec. 1, 19 students), and an afternoon section (Sec. 2, 18 students). There was a 90% response rate. The dynamics of each section were quite different, which is very evident in the survey results. As shown in Table 4, the 85% of the students reported that they made substantial or exceptional progress on the two essential learning objectives and no students reported no or slight progress. As for the second tier, important learning objectives, 62% and 67% of students reported they made substantial or exceptional progress on those, as well. These results provide evidence that students are reporting that they learned stuff! It is important to note that all four of the learning objectives received a higher rating on student's progress when compared to the average of all classes in the IDEA database (4.0, 3.9, 4.0, 3.7), all Mechanical Engineering classes in the IDEA database (4.1, 4.1, 4.0, 3.7), and all the classes at the home institution in the IDEA database (4.2, 4.2, 4.2, 3.9).

Learning Objective	Importance rating	Average Rating (5 pt. scale)			% of Students Rating 1 or 2			% of Students Rating 4 or 5		
		Sec. 1	Sec. 2	All	Sec. 1	Sec. 2	All	Sec. 1	Sec. 2	All
21. factual knowledge	Essential	4.5	4.3	4.4	0%	0%	0%	94%	75%	85%
22. fundamental principles	Essential	4.5	4.3	4.4	0%	0%	0%	94%	75%	85%
23. applying course material	Important	4.5	4.0	4.2	0%	6%	3%	88%	75%	62%
29. finding resources to solve problems	Important	4.1	4.1	4.1	6%	6%	6%	76%	56%	67%

Table 4 – A summary of the IDEA survey results for the essential and important learning objectives.

Going back to the three issues that were addressed for this course design,

(a) the course being offered outside of an engineering department,
(b) the extreme variability in the rate at which the students comprehend the material, and
(c) the frustration of new programmers, especially with debugging.

The IDEA survey results can be analyzed to determine if the students reported success in addressing these three issues. IDEA categorizes the answers to the survey questions into 5 different effective teaching methods and styles that can increase student learning. These include: stimulating student interest, fostering student collaboration, establishing rapport, encouraging student involvement, and structuring classroom experiences. These five teaching methods and styles can be tied directly to the three issues addressed in this course design.

Issue (a) was addressed in order to get first year engineering students interacting with engineering faculty earlier, to allow engineering students to learn to program in a practical programming language, and to allow engineering students to apply their programming skills to engineering problems. This issue ties most directly with the answers in the survey relating to stimulating student interest and structuring classroom experience. As summarized in Table 5, it was reported the following methods were effective and considered as strengths to retain: stimulated students intellectually beyond that required by most classes, demonstrated the importance of subject matter, made it clear how topics fit within the course, explained course material clearly and concisely, scheduled class work in ways which encouraged students to stay up-to-date, and provided timely and frequent feedback. Two teaching methods or styles were also effective and it was suggested to either retain the current use of the methods or consider increasing. These two were inspired students to set and achieve goals and gave tests that covered the most important points of the course. Finally, the teaching method that was found least effective and that should be increased is introduced stimulating ideas on the subject.

Issue (b) was addressed in order for each student to learn at a pace appropriate for their personal learning style and background. This issue ties most directly with the answers in the survey relating to fostering student collaboration and encouraging student involvement. These categories of teaching methods are much more challenging and therefore, the results showed less strengths in these areas. As show in Table 5, the strength to retain is gave projects, tests or assignments that required original or creative thinking. This is likely an crucial component to

variable student pacing. Two other teaching methods that were found effective were encouraged students to use multiple resources and involved students in "hands on" projects, which were both suggested to retain current use or consider increasing. In order to complete assignments students needed to self guide themselves through material, search the web, reference their textbook, ask their peers, and ask their instructor. And almost all the work done in the class was "hands on". Two teaching methods that will be increased include asked students to help each other understand concepts and related course material to real life situations. Additional evidence of issue (b) getting addressed can be found in Figure 15 and 16. Figure 15 shows a final distribution of course grades. The majority of students showed proficiency in the course outcomes. Figure 16 shows how the homework scores correlated with the final exam score. This shows that students who were able to complete homework assignments correctly performed better on the exams than students who did not complete homework or who did not complete homework correctly.

Issue (c) was addressed in order to help train students to problem solve and learn to debug in less frustrating environment. The teaching method that ties best to this issue is establishing rapport. As shown in in Table 5, the strengths to retain are found ways to help students answer their own questions, displayed personal interest in students and their learning, and encouraged student-faculty interaction outside of class. Explained the reasons for criticisms of students' academic performance was also considered an effective teaching method and it was suggested to retain current use of the method or consider increasing. Some of the methods in encouraging student involvement also played a role in addressing issue (c). Additionally, when asked to rate "students were comfortable asking questions in this instructor's class", 82% of students rated it 4 (more true than false) or 5 (definitely true) and no students rated it 1 (definitely false) or 2 (more false than true).

Other valuable feedback from the IDEA surveys was that students found that the technology used in the classrooms to be effective. When asked to rate "the instructor utilized current technology in the classroom in a way that made the course material more interesting", 88% of students rated it a 4 (more true than false) or 5 (definitely true) and no students rated it 1 (definitely false) or 2 (more false than true). The average rating was 4.5. This is an important aspect of a programming course. Lots of classroom technology was used including a clicker system, MATLAB software, document camera projector, student's personal laptops, and instructor laptop. If working with technology does not go smoothly in a class, it often does more harm than good.

Teaching Methods or Styles	Relevant Objectives	Average (5 pt. scale)	% of Students Rating 4 or 5	Suggested Action
Issue (a)				
Stimulating student interest				
13. Introduced stimulating ideas about the subject	21, 22, 23, 29	3.8/3.8/3.8	76/56/70	Consider Increasing
15. Inspired students to set and achieve goals which really challenged them.	21, 22, 23, 29	4.1/4.0/3.9	82/69/72	Retain current use of consider increasing
8. Stimulated students to intellectual effort beyond that required by most classes	21, 22, 23, 29	4.4/3.8/4.2	82/63/77	Strength to retain
4. Demonstrated the importance and significance of the subject matter	21, 22, 23	4.3/3.7/4.0	88/60/73	Strength to retain
Structuring classroom experience				
12. Gave tests, projects, etc. that covered the most important points of the course	21, 22	4.4/3.8/4.2	82/56/76	Retain current use of consider increasing
6. Made it clear how each topic fit into the course	21, 22, 23	4.4/3.5/4.1	100/56/78	Strength to retain
10. Explained course material clearly and concisely.	21, 22, 23	4.1/4.0/3.8	88/69/72	Strength to retain
3. Scheduled course work (class activities, tests, projects) in ways which encouraged students to stay up-to-date in their work		4.5/4.1/4.3	100/75/88	Strength to retain
17. Provided timely and frequent feedback on tests, reports, projects, etc. to help students improve.		4.3/3.9/4.1	88/69/79	Strength to retain
Issue (b)				
Fostering student collaboration				
18. Asked students to help each other understand ideas or concepts	29	4.2/3.6/3.9	76/50/64	Consider increasing
Encouraging student involvement				
11. Related course material to real life situations	23	3.8/3.5/3.7	65/56/61	Consider increasing
9. Encouraged students to use multiple resources (e.g. data banks, library holdings, outside experts) to improve understanding	29	3.89/3.4/3.8	71/56/70	Retain current use of consider increasing
14. Involved students in "hands on" projects such as research, case studies, or "real life" activities	29	3.8/3.9/3.6	65/69/61	Retain current use of consider increasing
19. Gave projects, tests, or assignments that required original or creative thinking	29	4.4/3.8/4.1	88/63/76	Strength to retain
Issue (c)				
Establishing rapport				
7. Explained the reasons for criticisms of students' academic performance	23, 29	3.7/4.1/3.6	71/69/70	Retain current use of consider increasing
2. Found ways to help students answer their own questions	21, 22, 23, 29	4.2/4.1/4.1	88/69/79	Strength to retain
1. Displayed a personal interest in students and their learning	23	4.5/3.6/4.3	94/69/82	Strength to retain
20. Encouraged student-faculty interaction outside of class (office visits, phone calls, emails, etc.)	29	4.6/3.9/4.3	94/75/85	Strength to retain

Table 5 – Summarizes IDEA survey results with respect to how three issues were addressed based on recommended teaching styles and methods determined effective by the IDEA center.

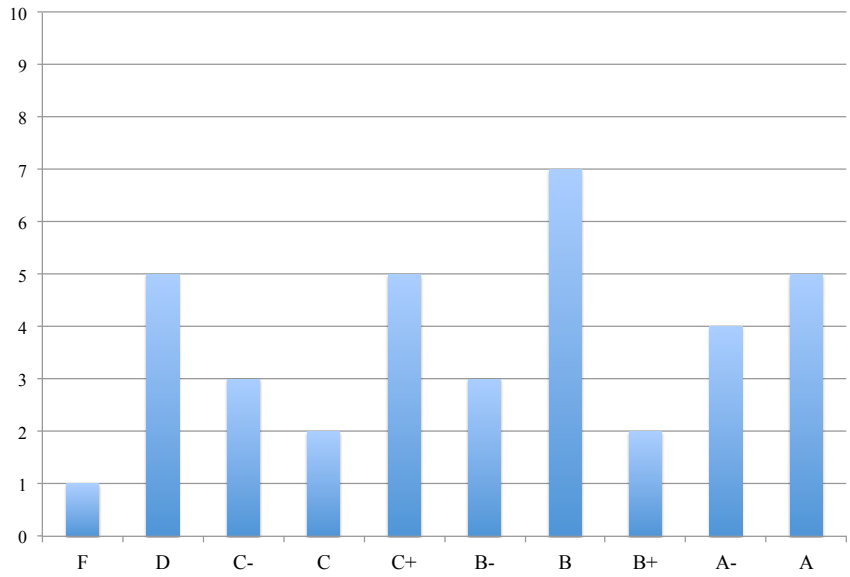

Figure 15 – Final grade distribution for the class.

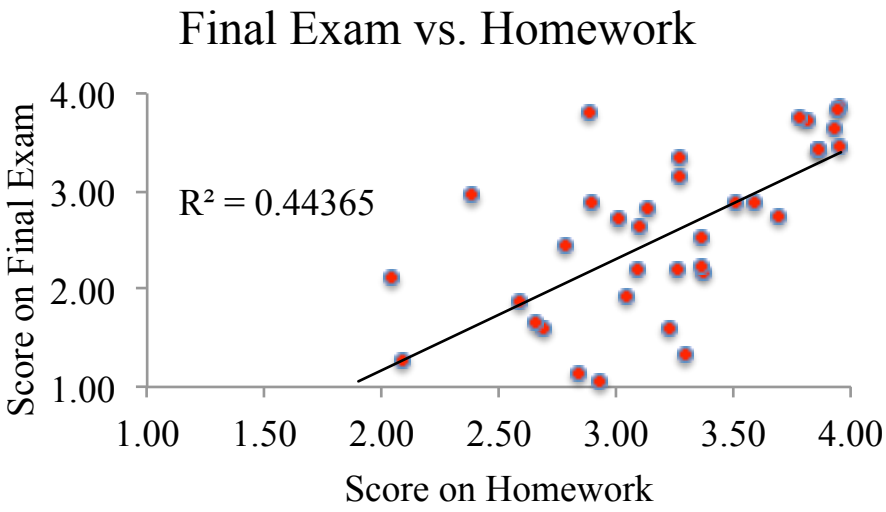

Figure 16 – Homework scores correlated to the final exam scores. The correlation coefficient squared is 0.44, which shows the two are moderately positively correlated.

Discussion and Conclusion

In summary, results show that the course design was effective. A few additional challenges are of particular interest to this design. First, the IDEA survey results show that students reported that they were not "asked to help each other understand ideas or concepts", which falls under the "fostering student collaboration" teach method (see Table 5). It was one of the poorest scoring qualities of the course, particularly by Section 2. This is likely due to the inherent unfortunate existence of programming and academic dishonesty. It is one of the most challenging components of teaching a course in programming. Letting students work together is important, but consistently and constantly requiring them to do their own work is extremely important for

effective learning. Unfortunately, this balance is difficult and students often don't understand what is "ok" and what is "not ok". The low scores in fostering student collaboration is most definitely due to this. In addition, this could also explain finding no correlation between working with others on programming assignments and scores on the midterm exam from Figure 11. Students are probably not comfortable reporting "how often they worked with others" on programming assignments because they are unsure what is allowed or feel they probably violated the academic honesty code in the syllabus. Due to the one-on-one nature of programming assignment assessment, a lot transparency regarding this issue was exposed and therefore, more easily avoided or mitigated.

The class is being taught for a second time in the Spring of 2014. There are many changes and additions to the course design that will be made based off of the results presented here. These changes are summarized as follows.

Longer class sessions. The course has been changed to a lab/lecture format, which is more indicative of the way it operates. The "work time" is similar to what is done is lab classes (computer or wet labs). Therefore, instead of meeting 2.5 hours per week, the course will meet 4 hours per week in two 2-hr sessions. Both the mid-semester survey and the IDEA student survey narrative comments indicated that many students wished they had more time in class for lecture and for work time. The second half the course, as it is usually, was more time consuming and could have benefitted greatly for the lengthening. This will also allow a lot more flexibility with more lengthy lectures when necessary or requested.

Notes used on exam. Many students commented in their IDEA student survey narrative comments that they did not find the toolboxes useful or did not understand the purpose of them. To address this, students will be allowed to only use tool boxes on exams. Therefore, they will need to organize and categorize all the content from each topic onto a topic crib sheet. This should clear up the purpose of the toolboxes and make them much more useful tools (for studying and test taking).

Humanitarian spin. In order to support the liberal arts goal of the university, the course will be themed around humanitarian engineering topics. Since the technical topics of the course (data analysis, statistics, curve fitting, interpolation, differentiation, integration, etc.) are not specific to a particular discipline within engineering, a overarching them of humanitarianism will tie everything together. Some of the programming assignments from Spring 2013 already fit this theme, such as analyzing climate data and reflecting on global warming. However, the Spring 2014 class will incorporate a humanitarian with all or most of the programming assignments. Some ideas include: looking at the UN's millennium goals, trash tracking, data on manufacturing produce and GMOs, focusing on the Water, Sanitation, and Hygiene UN cluster, data on food deserts, analyzing the Lorenz curve for different countries, data on factories in other countries, mining accidents, etc. The goal is for these to work well with the technical topics, be themed around humanitarianism, and also be strongly connected with the engineering discipline (i.e. an issue that might affect engineering in some way or that an engineer would be well suited to work on).

Class discussion and reflection. With the added class time, there will be more time to incorporate discussion on programming assignment results. For example, once they finish the assignment on analyzing climate data, there can be a discussion on the topic of global warming. This will also more time for reflection.

Pseudo-classroom flipping with video content. With the added class time and large number of students, an assessment method will be added. Students will be assigned to work in teams to produce educational, short videos highlighting what was learned in a particular topic. The videos will be open-ended to allow for maximum creativity. The students will post their videos on YouTube so that their peers can watch their videos outside of class and have an additional means to learn the material. This is in the sprit of classroom flipping, but with a twist.

References

[1] Siegel, C., Putting the Pieces Together: Linking Learning Outcomes, Assessment and Curriculum", Center for

[2] Farrell, J. J., R. S. Moog, J. N. Spencer, "A Guided Inquiry Chemistry Course." J. Chem. Educ., 1999, 76, 570-574

[3] Bergmann, J., A. Sams, "Flip Your Classroom: Reach Every Student in Every Class Every Day", International Society for Technology in Education, 2012

[4] Gonzalez, Virgilio, Eric Freudenthal. (2010). Work in Progress: Adoption of CCS0 Computational Methods and Circuit Analysis Techniques into an Introductory Programming Course for Electrical Engineers. American Society of Engineering Education Annual Conference and Exposition, Louisville, Kentucky.

[5] Karunaratne, Maddumage, "Learn MATLAB piggybacked onto C-programming", ASEE Conference Proceedings, 2013.

[6] Budny, D., Lund, L., Vipperman, J., & Patzer, J. L. I. I. I. (2002). Four steps to teaching C programming. In *Frontiers in Education, 2002. FIE 2002. 32nd Annual* (Vol. 2, pp. F1G-18). IEEE.

[7] Neuenhofer, Ansgar. (2009). Teaching and Learning Structural Engineering Analysis with MATLAB. American Society of Engineering Education Annual Conference and Exposition, Louisville, Kentucky.

[8] Soloway, Elliot. "Should we teach students to program?." *Communications of the ACM* 36, no. 10 (1993): 21-24.

[9] Fincher, Sally. "What are we doing when we teach programming?." In *Frontiers in Education Conference, 1999. FIE'99. 29th Annual*, vol. 1, pp. 12A4-1. IEEE, 1999.

[10] Ringenberg, Jeff, Marcial Lapp, Apoorva Bansal, Parth Shah. (2011) The Programming Performance Prophecies: Predicting Student Achievement in a First-Year Introductory Programming Course. American Society of Engineering Education Annual Conference and Exposition, Vancouver, B.C. Canada.

[11] Hamrick, Todd R., Robin A. M. Hensel. (2013). Putting the Fun in Programming Fundamentals – Robots Make Programs Tangible. *Proceedings of the American Society of Engineering Education Annual Conference*, Atlanta, Georgia.

[12] Sun, Wangping, Xian Sun. (2011). Teaching Computer Programming Skills to Engineering and Technology Students with a Modular Programming Strategy. American Society of Engineering Education Annual Conference and Exposition, Vancouver, B.C. Canada.

[13] Wiggins, G. P., & MCTIGHE, J. A. (2005). *Understanding by design*. ASCD.

[14] Accreditation Board for Engineering and Technology. (2014). Criteria for Accrediting Engineering Programs, 2012-2013. Retrieved from http://www.abet.org/DisplayTemplates/DocsHandbook.aspx?id=3143.

[15] Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). Taxonomy of educational objectives: Handbook I: Cognitive domain. *New York: David McKay*, *19*, 56.

[16] Individual Development and Educational Assessment (IDEA) Center. (2014). Interpretative Guide: IDEA Diagnostic Form Report. Retrieved from http://www.theideacenter.org/DiagnosticGuide