Journal of Engineering Education Transformations ,
Volume 28 , No.4, April 2015, ISSN 2349-2473, eISSN 2394-1707

Coding To Think: Teaching Algorithmic Thinking from Idea to Code

Joe Tranquillo
Bucknell University
377 Breakiron Engineering Building
Lewisburg, PA 17837 USA
e-mail:jvt002@bucknell.edu

Abstract: Engineering computing is a topic that nearly all engineering departments include in their curricula. Yet, the pedagogical goals of a computing course are necessarily split between code as a means of learning higher level math, code as a specific tool in design and research, and code as a way to learn algorithmic thinking. Learning more advanced applied math is typically learned through the traditional lecture/homework/test format, whereas learning the syntax of a particular language is most often taught through short programming assignments. This paper introduces Coding to Think as a way to teach algorithmic thinking that builds off of the Writing to Think movement in the Humanities. This technique is very well suited to long-term projects as it provides an opportunity to focus on deeper and more complex algorithmic thinking. The semester-long project presented is motivated by three guiding learning outcomes: 1) To program at a level of complexity that requires planning, iteration, encapsulation and documentation, 2) To move from Idea to Code (a phrase that is mentioned in class at least once a week) and 3) To articulate and put into practice the power of a computing language that can do more than a calculator or Excel. The seven project assignments that lead students from an initial idea to final code are detailed, as well as an assessment of outcomes, student and faculty comments, suggested improvements and adaptations and ABET assessment measures.

Joe Tranquillo
Bucknell University
377 Breakiron Engineering Building Lewisburg, PA 17837 USA
e-mail:jvt002@bucknell.edu

Keywords: Project Based Learning, Computational Instruction, Introductory Programming

1. Introduction

An introduction to engineering computing is a topic that is included in the curriculum of many engineering departments. Yet, the philosophical underpinnings and pedagogical goals of a computing course are often nebulous (Cordes 1997; Craig et al., 2008; Dunne et. al., 2005; Hambrusch 2009; O'Neill, 1996; Vergara, 2009; Wiebe, 2009). Should the course lean toward an advanced engineering mathematics course (e.g. linear algebra, splines, numerical methods), with programming as a means to an end (Hambrusch, 2009; Meyer and Jones, 2007, Miller and Winton 2004; Musante 2006; VANTH) ? Or should the course be designed to learn a specific computer language, such as MATLAB, as an example of an engineering tool (Clough, 2002; Clough et. al., 2001; Dunne et. al., 2005; Naraghi and Litkouhi, 2001; Shiavi and Brodersen, 2002; Thomassian et. al., 2007)? Alternatively, the course could be structured to teach algorithmic thought processes (Dunne et. al., 2005; Hambrusch, 2009; Musante, 2006; Von Lockette, 2006; Wiebe, 2009; Wing, 2008; Wing, 2006). No one way is best and any computing course should address all three to some extent. The implementation of a computing course, however, does need to be tailored to the objectives and backgrounds of the students. For example, the lecture-homework-test progression may be excellent at addressing an applied math objective, while short programming

assignments may address the learning of syntax. Here is present a semester-long project that has as its primary aim to address algorithmic thinking.

The paper is organized in the following way. Background is presented on how the project fits into an overall introduction to computing course. Next is a detailed outline ofseven project assignments followed by a description of a class-wide programming exercise called The Triangle Game. Student assessment is presented as well as the ABET assessments associated with the project. Lastly, recommendations are made for improvements on, and alternative implementations of, the project.

2. Background

The 15-week(1 semester) project was incorporated into a half-credit (two hours of formal lectures and one hour of recitation per week), required second-year undergraduate biomedical engineering course, Introduction to Engineering Computing. A typical undergraduate will enroll in 4.5-5 credits per semester, totaling approximately 20-25 hours of in-class instruction. The course wastaught five times by a previous instructor with the objective of introducing advanced engineering mathematics. The project was then included by the author to address a concern that students were not prepared for computing in later courses. The course has been taught with the project since that time five additional times.

The content of formal lectures was a mix of advanced applied math and practical lectures. Some unique lectures were delivered on topics such as the role of computing in the design process. A second lecture was delivered on how computing can be partnered with experiments and theory in the research process. Several cases were shown where computation either was, or was not, a good tool to move a research or design program forward. There were also short lectures on good programming habits such as how to write appropriate inline comments. Some lecture and recitation time was used for a "live" programming demonstration of The Triangle Game, as described below.

The syntax of MATLAB was learned through a text and individual weekly programming assignments. The instructor has published a short text that consisted of an introduction to common programming concepts (e.g. loops, conditionals, functions) and served as a self-guided tour of the MATLAB environment (Tranquillo, 2011). Programming assignments appear at the end of each chapter and counted for 15% of the final grade. To evaluate students on progress through the self-guided text, short quizzes were administered each week and counted toward 25% of the final grade. Professionalism (e.g. attendance, class conduct, meeting preparation) accounted for 10% of the grade, with the remaining 50% was allocated to the project.

3. Project Motivation

At the core of the semester-long project were two interrelated concepts. The first was Coding to Think, which parallels the movement of Writing to Think(Griffith, 1982, Read Write Think). Briefly, the philosophy of Writing to Think is to teach writing as a process by which the writer will organize, clarify and connect ideas. Communication to an audience is a secondary goal. The same can be said for the engineer and Coding to Think. The goal is to clarify constraints, parameters, and processes; critical thought processes vital to becoming a successful researcher or designer (Bundy, 2002; Craig et. al., 2008; Lunt and Ekstrom 2006; O'Neill, 1996; Vergara, 2009; Wiebe, 2009). Some previous work has been published on the idea of Coding to Think (Wing, 2008; Wing, 2006), but no publications were found on practical implementations of this theoretical idea. The one possible exception was the report of a secondary school environment where two teachers taught separate courses on Computing and Composition and made attempts to highlight the similarities between the two processes (Deek et. al., 2002).

The second concept underlying the project is how to move from Idea to Code. Here the focus was more on the practicalities of how to find and recognize a computationally tractable idea and then, beginning with a blank text file, write code that implements that idea. The idea could be a mathematical model, an analysis method or a graphical representation of a complex data set. While some simple ideas may be possible to implement in a few lines of code (similar to the weekly programming assignments), most original ideas require building code with a higher level of complexity and modularity. Ittakes time and individual practice to become comfortable writing code at this higher level. In other words, no amount of copying or modifying existing code or watching another code will suffice. As such, the project was designed to incorporate many aspects of active and problem based learning (Astrachan, 2004; Bowen,

2004; Gotfried, 2002; Said and Khan, 2004; Shiavi and Brodersen, 2002; Steadman, 2001)and spanned the entire semester. Although each of the references above presents various types of projects, none are the unique combination of being student-driven, long-term, and targeted at the learning of algorithmic thinking.

Semester-Long Project

The computing project presented below was designed to focus on the learning of algorithmic thinking. As a group project could too easily allow one person to perform the high-level algorithmic development leaving others to write only simple functions, each student completed every assignment on their own.

The parameters for the project were distributed on the first day of class with more specifics for each of the seven assignments given as handouts throughout the semester (Table 1).

Table 1: Due dates and percentages for each of the seven assignments

Assignment	Due Date	% of Project Grade
Initial Abstracts	3rd week	10
Background Presentation	6th week	15
Parametric Study	8th week	10
Draft of Code	10th week	15
Final Abstract	14th week	10
Final Presentation	15th week	20
Final Documentation	Finals Week	20

The project was motivated with the following text:

The central purpose of the semester-long project is to demonstrate that you can take an idea and transform it into working computer code. Your idea could build upon another's idea or it could be entirely original. The only constraints are that your specific idea cannot be published, the topic must have some biological or biomedical significance and the resulting computer program must include a variety of basic programming structures.

Although the focus of the project was on biomedical and biological systems, what is presented below could be easily adapted to other engineering disciplines. Students were encouraged to revisit the assignments listed below in an iterative fashion (similar to the design process) and to schedule additional meetings with the instructor as needed. Below is a description of each assignment.

Assignment 1: Initial Abstracts and Meeting

Each student was required to clearly and briefly communicate three distinct project ideas(1 paragraph each). Project parameters and topics were purposely vague to allow students the maximum flexibility in proposing projects. The only boundaries at this point in the semester were that the projects have some biological or biomedical significance. Students were encouraged to choose an idea that they would find interesting. Below are ideas provided to students on the first day of class:

- Agent Models (e.g. Slime Molds, Ants, Boids)
- Chemical Rate Equations
- Population Biology
- Evolutionary Game Theory
- Spread of Infectious Disease
- Cellular Gene Expression
- Ecological models
- Non-linear biological models
- Models at ccl.northwestern.edu/netlogo/

In general the above list would require students to perform a numerical simulation of a mathematical model. The list could be expanded (e.g. development of analysis methods, signal or image processing) but the author has found that it is more effective to start with simple mathematical models and then follow the interests of the students.

To help students generate more ideas, the instructor held a collaborative search session on the second day of class. Each student was asked to write down a few general thoughts about biological or biomedical systems that they found interesting. These ideas were shared with at least three other students. These groups of students were then required to help one another search through various books provided by the instructor, textbooks from previous courses and online sources (e.g.ccl.northwestern.edu/netlogo/). In this way, each student was not only looking for projects for themselves but also for other students.

The abstracts were a lead-in to an individual student-instructor meeting. At this meeting the instructor could gain a better sense of the proposed projects as well as help guide students toward a project that was of the appropriate scope. In preparation for

the meeting, students were requested to be able to answer the followingquestions:1) Is there a direction that you favor over the others? Why?2) Are there projects you are worried about pursuing? Why? They were also encouraged to bring any background information that they may have used as inspiration. For example, some students brought mathematical modelsthey had learned about in another class. Others brought websites or news articles. Students were also encouraged to prepare questions ahead of time and reserve time during the meeting to ask questions. They were reminded in class that a good strategy is to list questions and then arrange them in order of importance.

Assignment 2: Background Presentation

A few weeks after the initial meeting, all students were required to deliver a five minute project presentation to the class as well as a strategy for moving from idea to code. There were four purposes for the presentation. First, the deadline of the presentation required the clear definition of the project. As such, most students scheduled additional meetings with the instructor to finalize a project topic. Second, a background search had to be performed to demonstrate a deep understanding of the idea to be coded. As a five minute presentation is very short, a requirement was for each presentation to contain extensive notes and references in a notes section, beyond what would be said in the presentation. Third, all presentations were required to contain at least one slide on an approach to programming. This could include, but was not limited to, a discussion of the types of data structures and functions that would be required, a flow chart or an outline of a preliminary algorithm. The purpose for this requirement was to have students think early on about how they would implement the project idea. Fourth, the presentations allowed the rest of the class to understand the diversity and variety of topics and approaches. Students were instructed that an additional goal was to build excited about their project among their classmates.

Assignment 3: Parametric Study

The ability to perform controlled studies, where one variable is changed slowly with any resolution, while keeping all other variables constant, is one of the major strengths of numerical simulation. All students were required to identify at least two variables that would be varied over an appropriate range. The parametric study was introduced as a four-step process and documented in a two-page memo. The first step was to identify the parameters. There was a class discussion on how to choose parameters of a model that have real meaning. The second step was to determine an appropriate range over which each parameter should be varied as well as the resolution of the variation. The third step was to outline an analysis method to find the relevant properties of these data generated. The fourth step was to develop a way to summarize the analysis in a single figure (e.g. axes for each parameter varied with a color at each location indicating the results of analysis). Justification was needed (literature sources or a rationale) for all choices made.

Assignment 4: Draft of Code

- A few weeks after the parametric study design, a second student-instructor meeting was held at which time the student demonstrated progress made toward moving from idea to code. As outlined in a class handout, each project was required to integrate the following programming concepts into the project:

- Matrix-Vector Operations
- Loops
- Functions
- Conditional Logic
- Graphical and Data Output

As a baseline for "good" progress at this point in the project, students were given the following criteria for draft code:

- Define the key variables of interest
- Create data structures to hold the variables of
- interest
- Identify the key loops and conditionals needed
- Explain how the proposed algorithm will realize
- the project idea

It was at this meeting that the instructor could check that code was well documented with comments and readable by anyone with knowledge of MATLAB's programming keywords. It was also at this time that the instructor could point students to further resources. For example, students were encouraged to use MATLAB central (www.mathworks.com/matlabcentral) or online code sources with proper citations. As in Assignment 1, students were expected to show evidence that they had prepared for the meeting.

Assignment 5: Final Abstract

Two weeks before the end of the course each student turned in a one page abstract that summarized the project idea, methods, results, conclusion and future work. This abstract was included in a program booklet distributed at thefinal presentations (Assignment 6). More specific guidelines were distributed in a class handout along with examples of real computational abstracts from the Biomedical Engineering Society Conference.

Assignment 6: Final Presentation

During the last week of class each student delivered an eight minute talk followed by two minutes of questions. Minimum requirements for the presentation were:

- One slide reminding the class of the idea
- One algorithm slide with appropriate snippets of code or a flow chart
- One methods slide with a description of varied parameters and analysis methods
- One slide summarizing results
- One slide on conclusions, including the significance of results, major limitations and future directions

Although not required, it was suggested that students give a short live demonstration. As in Assignment 2, extensive notes were expected to supplement the presentation.

Assignment 7: Project Documentation

During finals week, all documents related to the projects were turned in electronically. These documents included all written work, all presentation slides (with notes) and a final working copy of all MATLAB code. In addition, aone-page users manual and one-page self-reflection were included. The objective of the users manual, code and presentations were to allow someone unfamiliar with the project (but familiar with MATLAB) to not only understand the project, but build upon it. The objective of the self-reflection was to identify the successful processes followed in finishing the project as well as processes that would be performed differently if the project were repeated.

5. Example Projects

Below are short summaries of three projects.

A biomechanics injury project simulated the effect of cyclic loading, as in athletic training. The simulation was derived from a differential equation based model of the lower extremities and included the option of dynamic spring constant that varied depending upon the duration and magnitude of load applied. A separate function integrated load over time in various segments to quantify the potential for injury. A user interface and built-in anthropomorphic table allowed a user to run simulations for a specific individual. The same user interface also allowed for the load cycles to be changed to simulate various sports and training schedules.

A population genetics project simulated ten genes on homologous chromosomes, two alleles for each gene, one on each chromosome. A fitness function took into account lethal alleles and carrying capacity. A reproduction function allowed two individuals with a fitness above a certain threshold to create a new individual with a new genotype. Also in the reproduction function was the possibility for mutation.

A disease propagation project simulated the spread of H1N1 on a hypothetical college population. A consultation with the director of health at our university yielded some experimental data of the spread of H1N1 at our institution and a few others. The model included simulations of the impact of various scenarios including no intervention, a large university-wide party, and the aggressive quarantine that was implemented at our university.

Live Coding: The Triangle Game

A concern of the instructor was that the project assignments, no matter how detailed, would not convey to the students how to move from idea to code. One of the more successful portions of the course was a series of "live" coding sessions, whereby the class cooperatively moved from idea to code. Although not explicitly related to the project, the instructor found that these demonstrations greatly helped make expectations clear as well as highlighted good algorithmic thought processes, inline commenting and debugging techniques. Below is a description of the Triangle Game, an invention of the author (Tranquillo, 2014). There are, however, a number of

excellent ideas in the references that could easily be adapted to be live coding demonstrations (Baibak and Agrawal, 2007; Gotfried, 2002; Lu et al., 2010; Maase, 2007; Miller and Winton, 2004; Musante, 2006; Myszka, 2006; Steadman, 2001).

The Triangle Game begins with everyone in the class walking around a large empty space. Each student secretly chooses two other students to track in the space. On a signal, each person follows one simple rule: attempt to make an equilateral triangle with your two targets. As each person attempts to make a triangle, the room becomes a dynamic whirlwind of repositioning. Sometimes the game will go on indefinitely. Other times, everyone will find a point where many stable triangles form and motion ceases. If the game is replayed, the results may be completely different. Surprisingly even without any barriers, the game typically remains bounded. As a classic example of an emergent system, no one player can cause the end result and the dynamics are governed by one very simple rule. As the game is very intuitive, it seems on the surface to be very easy to simulate on a computer.

The game was played and initial observations were listed as a group before any attempts were made to begin writing code. The instructor then guided students through a process of identifying variables (e.g. positions of each player), the geometry of finding equilateral triangles (e.g. use of basis vectors and rotations) and general program flow. The class then turned to translate the collective ideas into code. In this process it became clear that the dynamics of the game were far from simple. For example, there are in actuality two positions that might form an equilateral triangle. How should an individual student decide? Should the choice be random or the closest solution? Also the update rules can get complicated. For example, students discovered that the update could be synchronous (everyone moves at the same time at the end of an iteration) or asynchronous (pick a player at random and move only that player).They also found that it might not make sense to move directly to the goal on each iteration. Again this simple realization lead to suggestions from the class, such as to move a constant percentage of the way to the goal or with constant velocity. Decisions were made as a class and then the instructor took suggestions on how to transform the decision into code.

During the process of writing and running code, there were comparisons made to the initial observations. For example, the players in the first coded version did not remain bounded. At several points the Triangle Game was replayed to gain more insight. This back and forth between experiment and coding also served as a good demonstration of how to incorporate computing and experimental observation to learn about the deeper dynamics of a system. In the end the class had four different versions of the code, all of which reproduced some key observation. It was important that no version of the code perfectly reproduced the experimental observations, sparking a discussion about the validity of the simulations. It was at this point that the instructor was able to bring up that the reason for coding in this case was not to reproduce the experimental results but rather was an example of the power of Coding to Think. We collectively learned a great deal about the dynamics of the game by attempting to translate our ideas about the game into code. This deeper understanding then naturally leads to asking new, and more sophisticated, experimental questions.

A follow-up live coding session was conducted later in the course to build analysis tools for the results of the game. For example, it was easy to save the trajectories of each player and then show trends in the statistics. Again students were asked to invent analysis methods that would make sense. Although the class ran out of time to properly investigate the analysis of our results, our approach to analysis, as another example of Idea to Code, could be expanded in future iterations of the course.

6. Assessment

In this section, student numerical data is presented from one of the offerings of the course, as well as faculty observations and ABET criteria met by the project. The Bucknell University Institutional Review Board approved all data collection procedures. The final section offers improvements and modifications for instructors who might wish to adopt the project.

Student Observations

Student observations were obtained through a mix of numerical and written data from both informal questions (Table 2) and formal University Assessment questions (Table 3). It should be noted that the scores in Table 2 are on a 4 point Likert scale while those in Table 3 are on a 5 point Likert scale.

Table 2: Informal Student Evaluations (n=14) with a Likert Scale (1-Strongly Disagree, 2-Disagree, 3-Agree, 4-Strongly Agree)

Question	AVG	STD
Picking my own project helped me be more invested in the project	3.8	0.4
The individual nature of the project helped me learn more	3.5	0.4
The semester-long project was a good way for me to learn algorithmic thinking	3.5	0.8
The constraints on the project (i.e. loops, matrices) helped me focus my coding	3.2	0.5
The timing of assignments was helpful in staying on track	3.7	0.3
The project was a good way to bring together the concepts learned in class	3.5	0.8
I understand how matlab can go beyond what is possible with excel or a calculator	3.9	0.3
Electronic submission of work was helpful	3.7	0.4

Table 3: Formal Student Evaluations (n=14) with a Likert Scale (1-Strongly Disagree, 2-Disagree, 3-Neutral, 4-Agree, 5-Strongly Agree)

Question	AVG	STD
The open-ended project was a valuable part of this course	4.21	0.5
The project demonstrated the need for higher level algorithmic thinking	4.43	0.9
Material and hands-on skills learned in class sessions were helpful in staying on track	4.60	0.8
The sequence of assignments and deadlines were useful in staying on track	4.27	1.1

Below are select student responses to the question, "What was the best part about the project?"

"The open-ended project helped connect all of the material learned in class."

"The outcome: I can code in MATLAB"

"Choosing out own topics was the best thing to enable me to stay interested in the project".

"I loved how I was getting real results that could actually be applied to real life"

"Because we did all wildly different things people were willing to lend a hand to each other".

Below are select student responses to the question, "How can the project be improved?"

"It was very challenging to think about an algorithm on my own. I wish I had even more practice before the project".

"This project is a time sink where I can spend 6 hours of time and have nothing to show for my work. Maybe this is the way engineering is and the way computing is. Then this is also the way I should be graded".

"The project counts a lot toward the final grade. If it counted less we might be willing to take more risks".

"I'd like to spend more time in class just working on the coding in matlab. Taking notes didn't help much at all".

"More structure and possibly different guidelines individualized on each project".

Although not asked, a number of students commented on Coding to Think.

"I found that when I was going about my daily routines, I realized that they are really just while loops. I also started seeing the mundane decisions I make to be if-then statements".

"The night before everything clicked in my project was the night I dreamt about coding in matlab. It was scary."

A number of comments were also targeted at the Triangle Game demonstration.

"I hadn't coded before so an actual demo of the thought process and good coding techniques was really helpful".

"It exposed how real programming is completed. It dispelled the mystic [sic] that good programmers simply 'know what to do' and that there are many possible directions a large coding project can proceed in."

Faculty Observations

Through out the iterations of the course the instructor made notes on interactions with students in and out of class as well as on trends in the final projects. In general, programming courses may have one of the most widely distributed ranges of previous skills. While some students had been programming for a very long time, others have never learned to program. The instructor found that both populations struggled. The novice programmers struggled with mastering the basics of algorithmic thinking. Many did not appreciate that they would not learn to code by simply watching the instructor in class or copying code straight from the text. The more advanced

programmers struggled with the particulars of MATLAB, most especially the variable number of arguments to built-in functions. They also brought the preconception that they already knew how to program and were therefore frustrated that what should have been an easy class turned out to be a great deal of work.

In general final projects were above the level expected. Code was well-documented, organized, and met the overall curricular need to give students more experience with programming. Many projects included features not required. For example, nearly all projects allowed for some type of user interaction, either through command line inputs or a Graphical User Interface (GUI). Many students included some extra code to demonstrate how to run simple cases in the user's manual.

Students often mentioned the Triangle Game as the most helpful aspect of the course. As the instructor, the live programming demonstration provided an opportunity to reinforce the expectations for the project and it became a reference point for the remainder of the course. Students made the jump, with some prompting, to discuss how similar programming techniques could be used to model cells crawling on a scaffold, cells, organ and organism development and the spread of viruses and rumors on social networks. In conversations with students, the most important aspect of the live programming was that the instructor had not fully programmed the task ahead of time. Although this was largely true, the instructor did have a detailed algorithmic outline that could be referred to if the class stalled in developing ideas. It was also important to spread the demonstration across multiple sessions so that problems arising in one session could be answered in the next session. The only significant problem with the triangle game was that students initially expected the instructor to continue on to investigate all of the possible algorithmic options. This was taken as a sign that the students were truly engaged.

Despite the good qualities of the projects, there was ample room for improvement. The most significant problem was that the parametric study was not a natural fit for all projects. More flexibility could be built into this aspect of the project, especially if non-modeling projects are attempted. Although many of the projects clearly demonstrated a high level of algorithmic thinking, few projects showed a consideration for algorithm optimization. There were complaints that the project counted for 50% of the grade. After some questioning it came out that much of the source of this feeling was that the project was individual. All semester-long projects to this point in our curriculum are performed in a group. This was the first project where individuals where solely responsible for all work. Many of these problems could be addressed by making slight adjustments to the project assignments and more carefully introducing project topics. A selection of these possible modifications is below in the section on Recommendations for Implementation.

ABET Assessment

The ABET assessment for our introduction to Engineering Computing course was established prior to the adoption of the project. One of the secondary goals of the project, was to move all previous ABET evaluations to the project. Direct assessment was made for the following ABET outcomes

- 3a. an ability to apply knowledge of mathematics, science and applied sciences
- 3e. an ability to identify and solve applied science problems
- 3k.an ability to use the techniques, skills and modern scientific and technical tools necessary for professional practice.

Clearly there are a number of opportunities for assessment that were not taken but would follow naturally from the project (e.g. 3b, 3c, 3g, 3i, and 3j).

Recommendations for Implementation

There are a number of changes that may enhance the effectiveness of the project. First, it may help to add an explicit requirement of analysis and interpretation of results. This critical part of any project has historically been lacking in the final presentations and documentation. Although the author has added such a requirement to the final presentation, few students rise to the challenge. It may be that an analysis and interpretation assignment could be added between Assignments 4 and 5. Second, as noted above, there could be more flexibility included in the parametric study assignment. Some possible alternatives would be to give students the option to focus on code optimization, build a user interface or include more sophisticated error checking. Third, the project was the first time that students had to develop an algorithmically complex set of functions. It may be helpful, as pointed out by some students, to require an intermediate sized mini-project half-way through the semester. It may be that the code draft assignment could be restructured so that each student must have completed a sub-goal of the larger project by the time

of the meeting. Fourth, additional student comments revealed that the theoretical (e.g. applied math) section of the course seemed disconnected from the project and the overall goals of Coding to Think and Idea to Code. One solution would be for short live coding demonstrations (similar to the Triangle Game) to solve problems that require some knowledge of more advanced mathematics. For example, higher-order numerical integration techniques could be discussed in theory first and then used in a live coding demonstration. Lastly, as correctly pointed out in one of the student comments above, it was difficult to truly assess debugging efforts. Some possible recommendations are to ask students to comment on debugging efforts as part of the self-evaluation, keeping a coding log or turning in multiple code versions.

Because the course is a half credit, it is expected that students could dive more deeply into the project in a full credit course. For example, only one chapter of the text was assigned per week. It would certainly be possible in a full credit course to move more rapidly through the chapters, allowing time for a more thorough introduction to the various MATLAB toolboxes, a tighter link to applied mathematics, and more live coding sessions.

The instructor had (and expects to have in the future) small class sizes. It is not clear how this project will scale to larger class sizes, but it may be helpful to create project groups. A change to group projects would necessitate more specific guidelines to ensure that every member is taking part in generating ideas, developing algorithms, debugging, designing appropriate analysis and communicating results.

There are a number of ways in which the project could be slightly altered to fit into other types of courses. For example, in a full-credit course, the project might work well in combination with another course such as systems physiology. Although MATLAB (Mathworks) was used as a basis for the project, other high-level languages such as Python, Java or C++ would provide the same medium for implementing Coding to Think. In addition, the project could easily form the basis for a graduate class in engineering computing by simply altering the expectations. Even more broadly, the project could be modified for nearly any scientific discipline where computing plays an important role (e.g. physics, chemistry, biology).

A decision was made to focus the projects on mathematical modeling. This decision was based solely on the expertise of the faculty member and could be tailored to better fit the expertise of another instructor (e.g. signal or image processing, statistical analysis of biological data). Only one student deviated from a pure modeling project, producing anexcellent project to demonstrate how various levels and types of hearing loss affect the ability to recognize human-produced speech.

7. Conclusions

Computing and simulation are playing an increasingly important role in research and design (Craig et. al., 2008; Dunne et al., 2005; Lunt and Ekstrom, 2006; O'Neill, 1996; Wing, 2008). Although programming languages will come and go, the ability to think algorithmically is a skill that engineering undergraduates will need,not only in their first job, but throughout their40+ year career (Deek et. al., 2002; Dunne et al., 2005; Vergara, 2009; Wiebe, 2009). It is therefore important to teach algorithmic thinking as a critical thinking skill.

This paper proposed Coding to Think as an extension of Writing to Think, and Idea to Code as a practical way for students to learn Coding to Think (Wing, 2008; Wing 2006). The semester-long project presented here is a specific implementation of these concepts. Based upon student and faculty assessment and observations, the project is a start toward teaching students to think algorithmically. It will be necessary to follow up in future courses in the curriculum to fully realize the goal of algorithmic thinking (Craig et. al., 2008, Lunt and Ekstrom, 2006; Wiebe, 2009). In conclusion, the idea of Coding to Think could be implemented in many other ways and is an idea worthy of further investigation by others in the engineering education community.

References

Astrachan O. (2004) Non-competitive programming contest problems as the basis for just-in-time teaching. Proceedings of the Frontiers in Education Conference.

Baibak T and Agrawal R. (2007) Programming games to learn algorithms. Proceedings of the ASEE Conference.

Bowen J. (2004) Motivating civil engineering students to learn computer programming with a structural design project. Proceedings of the ASEE Conference.

Bundy D. (2002) Four steps to teaching C programming. Proceedings of the Frontiers in Education Conference.

Clough D. (2002) Teaching introductory computing to ChE students - A modern computing course with emphasis on problem solving and programming. Proceedings of the ASEE Conference.

Clough D, Chapra S and Huvard G. (2001) A change in approach to engineering computing for freshmen - Similar directions at three dissimilar institutions. Proceedings of the ASEE Conference.

Cordes D. (1997) Teaching an integrated first-year computing curriculum: Lessons learned. Proceedings of the Frontiers in Education Conference.

Craig A, Bullard L and Joines J. (2008) Computing across curricula. Proceedings of the ASEE Conference.

Deek F, Friedman R and Kim H. (2002) Computing and composition as an integrated subject in secondary school curriculum. Proceedings of the ASEE Conference.

Dunne B, Blauch A and Sterian A. (2005) The case for computer programming instruction for all engineering disciplines. Proceedings of the ASEE Conference.

Finlayson B. (2005) Introduction to chemical engineering computing. Proceedings of the ASEE Conference.

Gotfried B. (2002) Teaching computer programming effectively using active learning. Proceedings of the Frontiers in Education Conference.

Griffith M. Writing to think. (1982) National Writing Project Paper No. 4 National Endowment for the Humanities.

Hambrusch S. (2009) A multidisciplinary approach toward computational thinking for science majors. Proceedings of the SIGCSE Conference.

Lu Y, Zhu G and Koh C. (2010) Using the tetris game to teach computing. Proceedings of the ASEE Conference.

Lunt B and Ekstrom J. (2006) Changing times: The status of computing education in the United States. Proceedings of the ASEE Conference.

Maase E. (2007) Kangaroo thinking: Mathematics, modeling and engineering in introductory computer programming for engineers. Proceedings of the ASEE Conference.

Meyer G and Jones D. (2007) Advanced modeling in biological engineering using soft-computing. Proceedings of the ASEE Conference.

Miller D and Winton C. (2004) Botball kit for teaching engineering computing. Proceedings of the ASEE Conference.

Musante S. (2006) Strategies for teaching modeling to students. Bioscience 54: 4: 299.

Myszka D. (2006) Motivating students in an introduction to computing course by requiring animated solutions. Proceedings of the ASEE Conference.

Naraghi M and Litkouhi B. (2001) An effective approach for teaching computer programming to freshman engineering students. Proceedings of the ASEE Conference.

O'Neill R. (1996) Role of computing: Educator's Perspective. Proceedings of the ASEE Conference.

Read Write Think. Website [Online]. http://www.readwritethink.org.

Said H and Khan F. (2004) Toward using problem-based learning in teaching programming. Proceedings of the ASEE Conference.

Shiavi R and Brodersen A. (2002)Comparison of instructional modalities for a course - Introduction to computing. Proceedings of the Frontiers in Education Conference.

Steadman S. (2001) Enhancement of an introductory computing course with experiential and cooperative learning. Proceedings of the ASEE Conference.

Thomassian J, Kumazawa R and Kinnicutt P. (2007) A study of freshmen students' outlook to media based tutorials of Matlab/Java in computing for engineers. Proceedings of the ASEE Conference.

Tranquillo, J (2011) Matlab for Engineering and the Life Sciences, Morgan and Claypool, 2014.

Tranquillo, J (2014) Moving Analogies. Proceedings of the Venture Well OPEN Conference.

VANTH. Website [Online]. http://www.vanth.org .

Vergara C. (2009) Leveraging workforce needs to inform curricular change in computing education for engineering. Proceedings of the ASEE Conference.

Von Lockette P. (2006) Algorithmic thinking and MATLAB in computational material science. Proceedings of the ASEE Conference.

Wiebe E. (2009) Computing across curricula: The view of industry leaders. Proceedings of the ASEE Conference.

Wing J. (2008) Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A 366: 3717.

Wing J. (2006) Computational Thinking. Communications of the ACM 49: 3: 33.