Just-in-time Interactive Online Modules for Applied Engineering Computing

Lorena A. Barba and Adam M. Wickenheiser
Mechanical and Aerospace Engineering, George Washington University

Project abstract

We propose to develop and pilot a series of interactive online modules that teach programming **in context**, using a **high-level language**, and are used by students in a **just-in-time** fashion. Once the modules are deployed, students would complete them over the course of their degree (especially the first two years), as the programming skills are needed in the discipline courses. During Spring 2014, we will specifically target one sophomore and one junior-level course in Mechanical & Aerospace Engineering (MAE) to supplement with programming modules. The modules are intended to eventually *replace* the introduction to programming course and thus be *for-credit*. For example, one credit could be awarded on completion of, say, three modules. The motivation for this initiative is deep dissatisfaction of students with the current introduction to programming offering, as shown in the MAE Junior Survey 2013.

The initiative will be evaluated using Technology Acceptance measures in pre- and post-tests, with student focus groups. The groups will be divided into sophomores, juniors, and older students (seniors and 5th year B.S./M.S.) who can provide a retrospective assessment. The development and evaluation will be carried out during Spring 2014 and a report will be written in June 2014. The report will include a census of what other universities are doing to teach programming to engineering students, and a literature review of relevant studies. Subject to a successful evaluation, the idea is to continue in a follow-up project with implementation in Fall 2014, running the modules simultaneously with the traditional programming course, and allowing students to sign up for either option. New surveys using the Technology Acceptance model will be administered then.

Background and Motivation

The MAE Junior Survey, administered to all juniors in the program at the end of the Spring 2013 semester, made evident deep student dissatisfaction with the introduction to programming course. Some replies suggested that C programming should be eliminated from the curriculum, and many requested training in a high-level language (Matlab, in particular). On the other hand, faculty teaching various disciplinary courses often have an activity or assignment that requires programming and they find that students are not able to apply what they learned in the programming course. Programming skills are also increasingly required to conduct undergraduate research in engineering.

Unhappiness with the introduction to programming course is a common theme in engineering curricula. This is usually a **first-year undergraduate experience**, occurring before the applications and context are formally taught. This arrangement contributes to the frustration that can lead students to transfer out of engineering or drop out. Our pilot, if successful, would serve as a model to reform the way computing is taught in other engineering departments.

One of the known factors that contributes to student frustration in an introductory programming course is the use of a low-level language (like C, as here at GW). The argument for using C is often the assumption that

learning programming with a compiled language allows students to develop important skills, which may not be attained when learning with a high-level language. The evidence, however, points to the contrary. Mannila et al. (2006) analyzed a set of novice-written programs in Python, Java and C++. They found that, thanks to its simple syntax, much less programming errors were made in Python—an advantage when teaching novice programmers who get frustrated by syntax errors. There were also less logic errors, and the code was more structured (in part, thanks to the indentation requirements of Python). And when students transitioned to a more complex language, there was no handicap due to having learned in a simpler language. Our initiative is to develop interactive modules replacing introductory programming, **using both Python and the domain-specific Matlab product** (more on language choice below).

Another factor contributing to frustration is that students do not immediately see the usefulness of what they are learning (and often, struggling with) in a first programming course. Successful initiatives to improve introductory computer science often implement "programming in context." For example, the changes introduced in Harvey Mudd College to increase not only general retention but also student diversity included: replacing the programming language (from C to Python) and introducing applied problems with interesting contexts. Their success has been outstanding, on all fronts.[1]

A recent report of the National Research Council (2011) on pedagogical aspects of computational thinking expressed that *"... the power of computational thinking is best realized in conjunction with some domain-specific content."* Applying this philosophy, Magana, Falk & Reese (2013) created a freshman course on computation and programming for materials scientists and engineers at Johns Hopkins University, designed to be learner-centered. Their evaluation of this course found that it was successful in increasing students' perceptions of ability and their recognition of the importance of computing in their fields.

Responding to the concerns mentioned above, our initiative will develop a **discipline-based-learning** experience for introductory programming, which is delivered **just-in-time**. Instead of having a one-semester programming class, students will experience short modules which are available at all times, and will be prompted by their disciplinary course instructors to complete the modules that are useful for class activities or assignment in those courses. Each module will be an achievable and low-frustration learning experience, will be interactive and fully online, yet supported by walk-in sessions with learning assistants and a Q&A site (e.g., on Piazza).

The idea of offering short, just-in-time online modules was inspired by a new initiative at Stanford University called **JOLTs**, for "Just-in-time Online Learning Tools," launched in Summer 2013 as part of the technology ventures program.[2]

Proposal Objectives

We aim to develop and pilot interactive online modules that can replace an introductory programming course for engineering majors. The modules will be self-contained units that present programming in the context of a problem, and students will be guided to complete them during the course of their engineering degree. We estimate that about 9 modules can replace a 3-credit, one-semester course. **The modules will be self-paced, online, richly documented, and focused on mastery rather than grade.** They will be supported by learning assistants through online Q&A and office hours.

[1] See the NY Times article "Giving Women the Access Code," by Katie Hafner, April 2, 2012.
[2] See http://jolts.stanford.edu/math

During Spring 2014, we will specifically target one sophomore-level course (ApSc 2058 "Analytical Mechanics II") and one junior-level course (MAE 3134 "Linear System Dynamics") to supplement with programming modules. These courses in particular require multiple computational tools that the students historically have had insufficient practice with. Instead of interrupting the disciplinary curricula of these courses with lectures on programming, the proposed online modules will be used by students outside of class hours. Other courses in the MAE program could receive similar supplements after this pilot program is completed and evaluated (see Appendix 1).

The learning experience provided by the modules aims to develop not only programming skills but algorithmic thinking, in context. The aim is to *prepare students to use programming* in later classes to solve problems and develop confidence and computational literacy that will open doors for employment, undergraduate research or graduate study.

Format of the modules and learning experience

Each module will focus on an *applied problem* that needs to be solved with *computing as a tool*. It will include a short (5-min) motivation video, richly-formatted documentation (text, equations, and figures alternating with sample code snippets), and sub-goal labeled instructions (Margulieux et al., 2012). The students will be instructed to run embedded code snippets, extend or modify the sample code and obtain a solution to an achievable problem. The expected output of practice problems will be provided so the students get *immediate feedback*. As described below, the skills covered in each module will be developed in consultation with the course instructors such that the application of each skill is clear and the link to other coursework is obvious. A "dig deeper" section (à la TED-Ed) will include references to textbooks or links to online material where the students can learn more.

Success in the modules will be a requirement for completion. In other words, we adopt a *mastery-learning* approach: instead of assessing student learning at a set time and assigning a grade, students will work on each module until they achieve mastery of the problem and concepts it teaches. (The module assessment will then be marked with a "pass" grade.) Various innovative competency-based programs are emerging in which the tenet is mastery, not grades. One example is College for America, whose presentation literature reads that the *"primary form of assessment is completion of a task."* [3] We believe that programming skills are ideally assessed in this manner.

A necessary component of competency-based learning is strong student support. The format for support in this effort will be via online Q&A and walk-in tutoring opportunities (office hours). At the stage of this pilot, the funded graduate student assisting the PIs will hold the office hours, and the graduate teaching assistants of the supported courses will also be involved. They will be trained in the computing modules and, as part of their regular duties, they will hold office hours in support of their assigned courses, which will include assisting students with applying the programming tools in assignments given in those courses. Finally, the progress of participating students will be tracked, and a plan of intervention will be put in place when any student falls dangerously behind.

[3] See: slide 12 of the presentation PDF in:
http://collegeforamerica.org/site_images/Reduced_College_for_America_Public_Version.4.16.13.pdf

Evaluation of the initiative

We will apply the Technology Acceptance model, which was developed specifically for the study of computer-technology acceptance. It proposes that acceptance of a technological innovation is determined by two judgements about the technology: perceived usefulness and perceived ease of use. Application of this model to study the acceptance of educational technology by both students and faculty is well-documented (Teo, 2011).

Magana et al. (2013) applied the Technology Acceptance model to evaluate a curricular innovation to introduce computing to a materials science and engineering program, consisting of a new computing course, plus embedded computational modules in other courses. The focus of their evaluation was to determine students' acceptance of computation as a tool for their continued studies and future careers. We intend to follow their example.

During Spring 2014, and as the modules are being developed, we will call for volunteer students to test the modules (giving them, say, a $50 gift card). Following Magana et al. (2013), we will use a pre/post test with both Likert-survey and open-ended questions, intended to measure predictors of future behavior in relation to use of computing in future studies and careers. Questions will address **perceived ability** (to design an algorithm, to visualize data, etc.), utility, and **intention to use** in future coursework, projects, and in their careers (see some sample questions in Appendix 2). We will then obtain a composite score in each of these three categories. These students will also be given a survey at the end of the semester asking them to evaluate how well the modules prepared them to do the programming work required in the supported courses. Some of the questions in this survey will ask students to identify particularly difficult areas and how they would be better supported by the modules. At the end of the semester, we will also engage with the instructors of the disciplinary courses to ask their perceptions about student preparedness to apply computing for problem solving. Both the student and instructor feedback will be used to improve later versions of the modules, to create new modules to support the most difficult concepts, or to provide references to additional back-up materials.

We acknowledge that the Technology Acceptance model does not *directly* measure the learning that happens, but is rather an *indirect* measure—by interrogating students' perception of ability and utility of computing, we anticipate future behavior in how students might use computing in their studies and careers. Students' perception of ability is an important measure, however, and in the case of programming, it is particularly useful. If students are confident of their ability to use computing to solve problems, they are more likely to attempt it, and thus continue to get better at it.

Choice of language to teach introductory programming

The debate about programming languages is of historic proportions. But often in the debate the question of **suitability for teaching** novices is not in focus. Back in 1971, Wirth already noted that popularity is *not* a good reason for choosing a language for teaching. The fact that a language is widely used in industry has no relation to its suitability in helping a novice learn to think algorithmically. There are more important criteria. Quoting Bertrand Meyer (1993): *"A good teaching language should be unobtrusive, enabling students to devote their efforts to learning concepts not syntax."*

In Mannila & de Raadt (2006), the following criteria are offered for a language used in an introductory programming course:

- it was designed with teaching in mind (simple syntax, natural semantics)
- it can be used to apply physical analogies (provides multi-media capabilities)
- it offers a general framework (serving as a basis for learning other languages later)
- it promotes a new approach for teaching (augmented by principles, tools and libraries)

With the addition of other criteria involving the design environment (e.g., interactivity), support and availability, these authors draft a table of ratings for 11 popular languages (chosen on the basis of a census). The top languages for teaching according to this comparison are Python and Eiffel, followed closely by Java.

Matlab is not mentioned in the literature about languages for teaching because it is not a general-purpose programming language—it is a domain-specific language for linear algebra. The literature most often is aimed at CS or related majors, not engineering, where Matlab is popular. Several GW engineering faculty use Matlab, and students have the perception that they will use it in industry (we don't have evidence of this, or the contrary, at this time). For numerical computing, Matlab and Python with the numerical libraries are very similar: they both have simple and clear syntax, many built-in functions, are interactive and produce good media. The difference between the two is that Python is free and open-source, while Matlab is commercial and proprietary. But given the faculty preference at this time for Matlab, we propose to develop all the teaching materials in **both Python and Matlab**. Students will be free to choose for each module which one they want to use.

Activities and Budget

Development of the modules—We will hire a graduate student on an hourly basis to work with us in the identification of the context and problems for each module, the development of the code and documentation, and the administration of the student focus groups. The modules will be written on two platforms: (1) IPyhon Notebook[4], and (2) Matlab Publishing[5].

Budget for graduate student wages is $9,000, of which $3,000 is co-funded by the office of the Associate Dean of Research and Graduate Studies in SEAS. Requested from the TLC for this item is **$6,000**.

We will request assistance from the GW Instructional Technology Lab for the creation of 5-min videos to motivate and introduce the theory for each module. For digital-video editing services and website development, we estimate a cost of **$2,500**.

Evaluation with student focus groups—We will begin posting modules by end of March 2014, and evaluating them using student focus groups. We estimate 3 focus groups, with 10 students each, that evaluate 3 modules at a time. With a gift-card incentive for participation, the budget for this effort is 3x10x$50=**$1,500**.

Total budget requested: $10,000.

[4] See http://ipython.org/notebook.html
[5] See http://www.mathworks.com/academia/matlab-examples/

References

Magana, Alejandra J., Michael L. Falk and Michael J. Reese, Jr. (2013). **Introducing Discipline-Based Computing in Undergraduate Engineering Education**. ACM Trans. Comput. Educ. 9(4), Article 39 (April 2013), 22 pages.

National Research Council of the National Academies (2011). **Report of a Workshop on the Pedagogical Aspects of Computational Thinking**, Committee for the Workshops on Computational Thinking. http://www.nap.edu/catalog.php?record_id=13170

Magana, Alejandra J., Michael L. Falk, Mike Reese, Camilo Vieira (2013). **Materials Science Students' Perceptions and Usage Intentions of Computation**, 120th ASEE Annual Conference & Exposition, June 23–26, 2013. http://www.asee.org/public/conferences/20/papers/7450/view

Mannila, Linda and Michael de Raadt (2006). **An objective comparison of languages for teaching introductory programming**. In Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006 (Baltic Sea '06). ACM, New York, NY, USA, 32-37. DOI=10.1145/1315803.1315811 http://doi.acm.org/10.1145/1315803.1315811

Mannila, Linda, Mia Peltomäki, Tapio Salakoski (2006). **What About a Simple Language? Analyzing the Difficulties in Learning to Program.** Computer Science Education 16(3): 211–228.

Margulieux, Lauren E., Mark Guzdial, and Richard Catrambone (2012). **Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications**. In Proceedings of the ninth annual international conference on International computing education research (ICER '12). ACM, New York, NY, USA, 71-78. http://doi.acm.org/10.1145/2361276.2361291

Meyer, Bertrand. **Towards an object-oriented curriculum.** In Proceedings of the 11th International TOOLS Conference, pp 585–594, Prentice Hall, 1993.

Teo, Timothy (ed.). **Technology Acceptance in Education**, Sense Publishers (2011) https://www.sensepublishers.com/media/1035-technology-acceptance-in-education.pdf

Wirth, Niklaus. **The programming language Pascal**. Acta Informatica 1: 35–63 (1971)

Appendix 1

Client MAE Courses

The following courses are identified as targets for being served by the interactive computing modules. We will interact with the faculty teaching (or having taught in the past) these courses to develop appropriate problem-based computational modules. We have also identified a list of fundamental mathematical and scientific computing concepts to be included in the introductory modules.

- Fundamental Concepts (including material covered in MATH 1231 and 1232 "Single Variable Calculus 1 and 2" and MATH 2184 "Linear Algebra")
    1. Symbolic differentiation and integration (single and multi-variable)
    2. Finding the minimum or maximum of a function with a given formula, locating crossings of the x- or y-axis, doing the same with a set of data
    3. Plotting a function and approximating its derivative and integral
    4. Numerical techniques for partial differentiation and multiple integration
    5. Plotting curves, surfaces, and vector fields in 2-D and 3-D, contour plots, projections
    6. Plotting trajectories in vector fields
    7. Populating vectors and matrices programmatically, combining arrays, row reduction
    8. Solving systems of equations using matrices
    9. Computing eigenvalues and eigenvectors
    10. Applying linear transformations to sets of data, for example rotating or stretching an image
- ApSc 2058 "Analytical Mechanics II"
  This course covers the dynamics of particles, rigid bodies, and interconnected systems. The objectives of this course are for students to be able to identify the forces and moments acting on a body and to describe its motion under these conditions. Students solve the resulting equations of motion analytically when possible and compare to computational solutions found using standard techniques such as Runge-Kutta methods.
  Programming modules would complement the course in the following areas:
    1. Symbolic solution of systems of ODEs
    2. Numerical solution of systems of ODEs using built-in solvers such as Matlab's ode45
    3. Plotting analytical and numerical solutions side-by-side and quantitatively analyzing the differences
    4. Estimating velocity and acceleration from position data (or vice versa)
    5. Creating animations to visualize the motion of bodies or systems of bodies, tracking the motion of specific points on a body such as the center of mass
- ApSc 3115 "Engineering Analysis III"
  This course covers the basics of probability and statistics. Emphasis is placed on engineering applications such as correlation, curve fitting, and hypothesis testing.
  Programming modules would complement the course in the following areas:
    1. Calculating statistical measures such as mean, standard deviation, and cross-correlation of data sets
    2. Importing (and potentially reformatting) data sets from sources such as data acquisition systems, sensors, and simulation output
    3. Simulating random events, generating white or colored noise data sets with a given probability distribution
    4. Curve fitting using single or piece-wise functions, computing correlation and regression statistics
    5. Computing confidence intervals, plotting data with statistical information
    6. Computing the frequency content of measured data and applying digital filters
- MAE 3134 "Linear System Dynamics"
  This course covers the modeling of mechanical, electrical, and fluid systems using transforms and state space. Students determine the dynamic response of systems by analyzing their mathematical models and simulating them.
  Programming modules would complement the course in the following areas:
    1. Symbolically computing the Laplace and inverse Laplace transforms of functions, partial fraction expansion

2. Plotting the free and forced responses of systems, inferring information about the system from the response plots
3. Converting between transfer functions and state space models
4. Plotting Bode plots of the frequency response of a system and comparing to analytic approximations

- MAE 4182 "Electromechanical Control System Design"
  This course covers classical control theory, including root locus and frequency response methods of control design. Students use Matlab/Simulink to simulate the response of systems to various inputs before and after controllers are integrated.
  Programming modules would complement the course in the following areas:
  1. Plotting the response of systems to inputs, before and after control is applied, comparing these responses in terms of standard performance metrics such as settling time
  2. Plotting the root locus of a system, using root locus techniques for control design
  3. Plotting the Bode and polar plots of a system, using frequency response techniques for control design
  4. Using Simulink to combine multiple sub-systems and analyze the overall performance of the system

Appendix 2

Example survey questions

From Magana, Falk & Reese (2013), some example questions in applying the Technology Acceptance model to the programming models are (answered in a Likert scale):

1. I have the ability to design an algorithm
2. I have the ability to write a computer program
3. I have the ability to use a computer to solve a set of linear equations
4. I have the ability to visualize data using a computer
5. I have the ability to create a computer representation of [NAME SYSTEM]
6. I have the ability to numerically solve an initial-value problem
7. I have the ability to implement a numerical model based on a simple partial differential equation
8. I feel computation will be useful in my studies
9. I feel computation will be useful in my career
10. I intent to seek courses that will allow me to increase my knowledge about computation
11. I intend to use computation in my future career

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

November 2, 2013

The Office of Teaching & Learning
Grants for High Impact Teaching and Learning Practices

Dear Members of the Selection Committee:

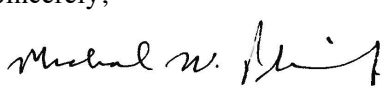
On behalf of the Department of Mechanical & Aerospace Engineering, it is my pleasure to enthusiastically endorse and strongly support Profs. Lorena Barba and Adam Wickenheiser's proposal entitled *Just-in-time Interactive Online Modules for Applied Engineering Computing*. The proposed work is centered on developing and piloting a series of interactive, online modules on engineering programming skills that will complement disciplinary courses in our program. The educational plans outlined in the proposal are well integrated with the goals of the MAE Department and the George Washington University Strategic Plan. As Chair, I am personally committed to provide guidance to the Profs. Barba and Wickenheiser, and to fully support this pilot program.

I have read their proposal, and I eagerly back Prof. Barba and Wickenheiser's plan to develop online programming modules to complement several of our required undergraduate courses. This proposal directly addresses repeated concerns expressed by our students (as reported in junior and senior surveys) about limitations of the current programming course and the lack of formal instruction in high-level, engineering-specific programming skills in a language such as Python and MATLAB. Prof. Barba and Wickenheiser will work directly with several of our faculty who teach disciplinary courses that will benefit directly from supplemental programming instruction. Through focus groups, they will assess the success of this pilot by applying a Technology Acceptance model to determine the students' acceptance of the technology and intended future use of the programming skills learned. If this pilot proves successful, they will also investigate how our current programming course might be replaced by a sequence of online modules.

In addition to its profound curricular contributions, the proposed education program will serve as an excellent training vehicle for undergraduate research that involves programming guided by MAE faculty. Profs. Barba and Wickenheiser and the MAE Department will make a concerted effort to involve women and underrepresented minority students in this project.

In summary, I believe that if selected for an award from the Office of Teaching & Learning, Profs. Lorena Barba and Adam Wickenheiser will be a credit to the Teaching & Learning Collaborative and will continue to develop into leading, innovative educators. I strongly endorse their proposal and commit wholeheartedly on behalf of the MAE department to provide strong support and guidance to ensure success in attaining the ambitious goals described in their proposal.

Sincerely,



Michael W. Plesniak
Professor and Chair
Mechanical & Aerospace Engineering