Work In Progress: From Scratch - The Design of a First-Year Engineering Programming Course

Dr. Philip Reid Brown, Rutgers, The State University of New Jersey

Philip Brown is an Assistant Teaching Professor in Undergraduate Education at Rutgers University in the School of Engineering. Philip received his PhD from the Department of Engineering Education at Virgnia Tech. His research interests include the use of motivation, cognition and learning theories in engineering education research and practice, pedagogy for programming courses, and better understanding student perspectives in engineering programs.

Work In Progress: The Design of a First-Year Engineering Programming Course

Abstract

This work in progress study concerns the design and implementation of a first-year programming course for engineering students at a large public university in the Mid-Atlantic United States. Mid-Atlantic University (MAU) accepts approximately 800 first-year engineering students annually, and has an enrollment of approximately 1200 students in its fall and spring Introductory Programming Class (IPC), taught in MATLAB. The IPC is currently under redesign through the process of Backward Design[1]. The research around this redesign attempts to answer the following question: How can the implementation of non-traditional pedagogy be used to increase self-efficacy, motivation, learning and retention in an introductory programming class? This redesign is being implemented in three phases. The first phase, which has been completed, emphasized the development of course outcomes and the implementation of active learning activities in lectures. The second phase, which is still ongoing, is emphasizing the implementation of supplemental online resources, a non-traditional digital textbook format, and the addition of a project to the course to give students more autonomy over their learning. The final phase, which has yet to come, will be a trial division of the class into two courses, one for students with prior programming experience, and one for students who are new to programming. This paper primarily discusses the implementation of and results from the first phase. Preliminary results show that students in Phase 1 withdrew from the course at a lower rate than in previous semesters. However, the continuing gap in motivation and self-efficacy between students who have programmed before and students who have not in both motivation and self-efficacy suggests that additional redesign is still needed.

Introduction

This work in progress study concerns the design and implementation of a first-year programming course for engineering students at a large public university in the Mid-Atlantic United States. Mid-Atlantic University (MAU) accepts approximately 800 first-year engineering students annually, and has an enrollment of approximately 1200 students in its fall and spring Introductory Programming Class (IPC), taught in MATLAB. The purpose of this redesign is to develop an appropriate set of course outcomes related to student learning and motivation, and build the course around those outcomes. A secondary purpose of this redesign is to improve student persistence from IPC towards later engineering courses, as IPC is a required first-year course at MAU.

This redesign ongoing, and is being implemented in three phases. The first phase, which has been completed, emphasized the development of course outcomes and the implementation of active learning activities in lectures. The second phase, which is still ongoing, is emphasizing the implementation of supplemental online resources, a non-traditional digital textbook format, and the addition of a project to the course to give students more autonomy over their learning. The final phase, which has yet to come, will be a trial division of the class into two courses, one for students with prior programming experience, and one for students who are new to programming. This paper primarily discusses the implementation of and results from the first phase.

While there have been recent calls to strengthen computer science and programming education in K-12 curricula [2-5], secondary programming education is still in its infancy. As a result, there are still a large number of first-year engineering students with little or no prior programming experience. At MAU, for example, of the students who responded to a course survey for IPC (224), a little more than half (114) had any kind of programming experience. Traditionally, students from under-represented groups have an even lower level of programming preparation. For instance, only 22 of 57 female students, who responded to

the IPC survey had programmed before. This lack of pre-college preparation contrasts with the need for programming skills. At MAU, most engineering departments have parts of curricula that either focus on computer programming or require students to have computer programming skills as a prerequisite. When student preparation is contrasted with the high demand for programming skills across engineering disciplines, the need for informed pedagogy in introductory programming courses was apparent.

This led MAU to consider a complete redesign of IPC, with a gradual rollout of new materials and assessments. The overall purpose of this redesign was to improve student learning and motivation outcomes related to programming, with a secondary goal of improving course persistence. We first chose a conceptual framework for the redesign, in Backward Design [1], a widely used framework for the design and assessment of academic courses. To assess motivation, we adopted the theoretical frameworks of Self-Efficacy [6-8] and the MUSIC Model of Academic Motivation [9] (See Table 2). The former is a proven predictor of learning and persistence in academics [10-13], the latter as it has been shown to be a practical, reliable way of interpreting and measuring constructs of course-level motivation that relate to learning[14-18]. The following sections are an overview of both frameworks and how they relate to the redesign of the IPC

Course Design Framework

Backward Design [1] is a widely used framework for course development and assessment of courses and curricula. It has three main steps:

1. Identify desired results: Come up with high-level course outcomes.
2. Determine acceptable evidence: Draft demonstratable objectives and assessments that measure them.
3. Plan learning experiences and instruction: Come up with coursework and course-related interactions that guide students towards demonstratable objectives.

Once the process is complete, best practice is to use the results of assessments to update course outcomes and objectives, and continue the process of design.

Using Backward Design as a framework, we broke the redesign of IPC up into three distinct phases, with each phase taking place in succeeding semesters. In the first phase of redesign, we focused on rewriting course outcomes and redesigning the materials used for the large lecture portion of the course (See Table 1, column 2). This phase was recently completed. In Phase 2, which is ongoing, we are implementing additional materials to give students more freedom and control over their learning (See Table 1, column 3). Phase 3, which is forthcoming and still in development, will focus on the possibility of splitting the class into two classes for those with and without programming experience. As Phase 3 is still in formulation, it is not included in Table 1. As mentioned above, this paper focuses mainly on work performed for Phase 1.

Figure 1 shows the relationships between all of the components of Phase 1. The solid line shows the group of objectives, evidence and classroom activities that related to learning subject matter in the course. The dashed line shows the group of objectives, evidence and classroom activities that had to do with motivation and self-efficacy. The overall research question for this study is: **How can the implementation of non-traditional pedagogy be used to increase self-efficacy, motivation, learning and retention in an introductory programming class?** Data collected at the end of Phase 1 helps to answer the self-efficacy, motivation and retention portions of that question. As the course content was altered significantly from previous semesters, no valid comparison of learning was available for the course before and after redesign.

Table 1: Alignment of Backward Design and IPC Development Phases

Backward Design Step	**Course Development from Phase 1 (Completed)**	**Course Development from Phase 2 (Ongoing)**
Identify desired results	Develop course outcome list grounded in needs of students and future courses (Appendix A) Increase student engagement in class Increase student self-efficacy for programming Increase retention in IPC	Increase student autonomy over learning in class
Determine acceptable evidence	Develop new pool of homework and exam questions Adapt and develop survey instrument	Adapt some high-level course outcomes to standards-based project deliverables Survey comparison of cohort before and after implementation **(Ongoing)** Interview students **(In development)**
Plan learning experiences and instruction	Develop lecture and recitation activities around homework and exam question topics Include interactive components in lecture	Implement project component of course Adapt some homework questions to instant-feedback online homework Implement online, interactive text
Cycle back and update	Compare withdrawal and failure rates with previous semesters. Analyze survey data **(Ongoing)** Interview students and faculty **(In development)**	Continued use of survey

Learning Outcomes

Desired Results

Course learning outcomes, shown in Appendix A, were determined by informally consulting faculty and students in different engineering departments in MAU, and were developed with the intent of helping students reach a practical knowledge of the essential elements of computer programming and their

implementation in MATLAB. Future phases will include more rigorous interviews with students and faculty in order to update these objectives.

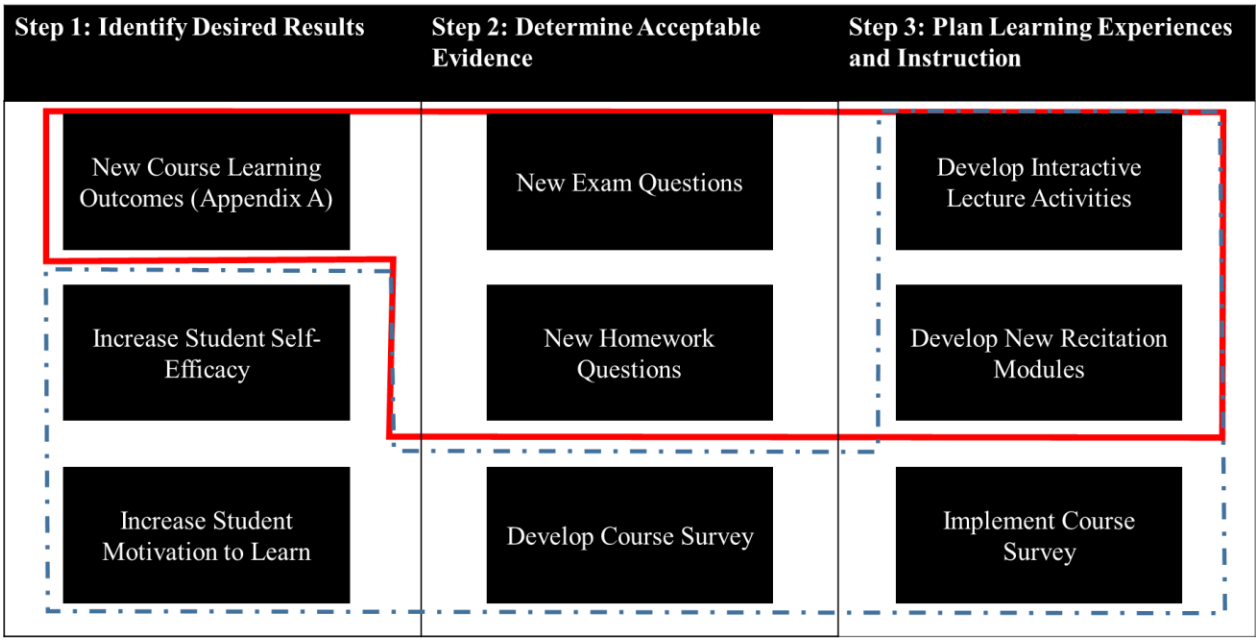

Figure 1: Backward Design of Phase 1

Acceptable Evidence

Once learning objectives were set, new exam and homework assessments were created to tie directly to those objectives. Care was taken to *not* extend the subject matter of assessments beyond topics from high-level learning objectives, even if some course instruction (i.e. portions of lectures) occasionally stretched past those outcomes to demonstrate the usefulness of some subject matter. For instance, though a later lecture showed how simple programming concepts could be used to model a cochlear implant, students were not expected to be able to demonstrate how such a model would work, as such an assessment would be outside of course objectives.

Learning Experiences

Once homework and/or exam questions were created, lectures and recitation modules were developed to tie directly to them. Lectures were in large groups of approximately 300 students, while recitations were in smaller groups of approximately 40 students. Both lectures and recitations were approximately 40 minutes long. Lectures generally covered a swath of material related to a set of learning outcomes. In the following recitation, students would explore the concepts from the previous recitation in more detail through hands on programming activities, facilitated by both graduate and undergraduate teaching assistants. Students would then have to complete homework assignments based on recitation modules.

Motivation and Self-Efficacy Outcomes

Desired Results

The development of this course was also informed by motivation and self-efficacy theory, and high-level course outcomes were set to increase both student motivation in the course and their self-efficacy as a programmer. Motivation was measured using the five constructs of the MUSIC Model of Academic

Motivation[9]: Empowerment, Usefulness, Success, Interest, and Caring. These constructs are defined in Table 2.

Table 2: The MUSIC Model

MUSIC Letter	Name	Definition
M	Empowerment	How in control a student feels about his or her own learning experience.
U	Usefulness	How useful a student thinks course material is to them.
S	Success	The student's belief in their ability to do well in the course.
I	Interest	How fun or interesting course material is to the student.
C	Caring	Whether the student feels that course instructors are empathetic towards how they experience the course

The constructs of the MUSIC model are geared towards course-level motivation. Collectively, they help to tell a story about how motivated a student is to learn in a course, and the factors that contribute to that motivation.

In addition to course-level motivation, we were also interested in knowing more about students' self-efficacy to program. Self-efficacy is one's belief in their ability in a certain task [8]. While the Success construct in MUSIC is similar to self-efficacy, in practice it is directed towards whether a student thinks they will do well in a course. Self-efficacy towards programming is *not necessarily* the same as a student's belief that they will succeed in a programming course. As such, we thought it was important to set a course objective to increase students' self-efficacy to program.

Acceptable Evidence

To measure motivation and self-efficacy, we adapted previously created instruments for each construct. Those surveys are detailed in the Methods section.

Learning Experiences

Motivation and self-efficacy outcomes influenced instructional methods in IPC's Phase 1 redesign. To reduce the chance that students with no programming experience would experience a quick decrease in self-efficacy at the beginning of the course, new concepts were delivered very slowly at the beginning of the course. Students who lack "success" experiences when learning a new subject will be more likely to have a low self-efficacy for that subject matter, and will be more likely to perform poorly on assessments, or withdraw from a course[8, 11, 12].

Additionally, lecture materials were designed to engage students more often, with the intent of increasing their motivation (via the MUSIC constructs). Instead of 80 minutes of instruction, each lecture consists of modules of approximately 15 minutes of instruction, interspersed with one or more interactive "clicker" quizzes, and one or more hands-on activity Students were encouraged to bring laptops to class and work in groups on these activities. For example, during a lecture about loops in programming, students were given instructions on how to place breakpoints in MATLAB, and then instructed to step through an example program containing a "for" loop in order to see the behavior of a loop as the program iterates. This design was intended to stop the introduction of new material before students experience fatigue, while providing them with the opportunity to use newly learned knowledge more quickly.

Methods

Participants

There were 596 students who took IPC during Phase 1 of the redesign. Almost all of the students who took the course were engineering students, either first-year students or transfer students, with a very small number of non-engineering students. We were not able to track the exact numbers of first-year and transfer students. We were, however, able to track the number of students who did not persist beyond IPC, due to withdrawing or receiving an inadequate grade in the course for Phase 1 and the two previous semesters of the course prior to Phase 1(See Table3). Students in Phase 1 were also given an end of course survey to measure motivation and self-efficacy. A total of 224 students responded to that survey.

Survey Instrument

To assess motivation and self-efficacy, surveys for both the MUSIC model and self-efficacy were adapted. For the MUSIC Model, Jones' questionnaire [19] for measuring each construct in a classroom setting was adapted verbatim. This questionnaire was designed to be used with any course, and consisted of 26 Likert scale items. To assess self-efficacy, the scale created by Brown and Matusovich was modified to measure programming self-efficacy [20]. The scale consisted of 5 Likert scale items measuring a single construct. Items from both adapted instruments used a 7-point Likert scale from Strongly Disagree (1) to Strongly Agree (7). Both Jones and Brown and Matusovich found both of these surveys to be valid and reliable. For detailed discussion of survey validity and reliability, please see each cited source. Students were also asked demographic questions, and whether they had any prior programming experience was prior to taking IPC.

Results

Table 3 shows a comparison of students who had to retake the course to continue in the engineering curriculum at MAU between Phase 1 of the course redesign and the previous two semesters. Phase 1 saw a 15-20% decrease in the number of students who were not able to persist through to future engineering courses from the previous two semesters.

Table 3: Comparison of Persistence Before and After Redesign

Semester	Total Students Who Had to Retake Course	Course Enrollment	Percentage
Phase 1	34	596	5.7%
Semester A*	123	573	21.5%
Semester B*	135	530	25.5%
*Two semesters prior to redesign			

A total of 224 students responded to the end of semester survey. There were no significant differences seen between students based on the demographics of gender, race, or ethnicity ($p = 0.05$. unpaired t-test for gender, one-way ANOVA for race and ethnicity). However, there were significant differences seen between students who had programmed before IPC and those who had not (unpaired t-test, $p = 0.05$, see Table 4). In interpreting Table 4, note that a score of 1 is low, a score of 4 is neutral, and a score of 7 is high. Non-programmers scored lower on both programming self-efficacy and belief in success in the course. Note that, across all students, students were generally motivated in the course (through each of the MUSIC constructs), while having a slightly lower (but still generally neutral to high) score for their self-efficacy to program.

Discussion

Phase 1 of the redesign of the IPC at MAU produced two meaningful findings. The first is that it was possible to increase the persistence rate of students in the IPC from previous implementations of the course. The second was that there are significant differences in IPC between students who have and have not programmed before. Each of these findings has implications for the future of the course.

Table 4: Comparison of Students With and Without Prior Programming Experience, Mean Construct Scores

	PSE	M	U	S	I	C
No Experience (110)	4.19*	4.96	5.01	4.90*	4.65	5.77
Prior Experience (114)	5.26*	5.63	5.55	6.07*	5.34	6.05
Total (224)	4.73	5.31	5.25	5.50	5.00	5.91

*Significantly Different (Unpaired t-test, $p < 0.05$)
Scales go from 1(Low) to 7(High)
PSE: Programming self-efficacy
M: Empowerment
U: Usefulness
S: Success
I: Interest
C: Caring

The large decrease in the number of students who failed or withdrew from IPC has a few possible explanations . One possibility is that, but slowing the introduction of course material and by taking care to not overextend assessments beyond course outcomes, it was simply easier to pass the redesigned IPC than the old versions. The other possibility is that, by carefully matching course materials to assessments, and assessments to outcomes, the redesigned IPC was better at facilitating students' learning experience. These possibilities are not mutually exclusive. Indeed, we believe that a course being "easier" is a natural consequence of re-evaluating course outcomes, and designing a course through Backward Design. When student learning is facilitated, the course will be "easier" to pass.

This does not mean that the course redesign for IPC has met its goal. On the contrary, the finding that students with no previous programming experience were less motivated and had less self-efficacy for programming after taking this course (as seen in the scores in Table 4) shows that there is work to be done in bridging the gap between students with more preparation and those with less preparation. While this finding is somewhat contradictory to studies such as Bergen et al. [21], which found that prior programming experience does not affect performance in a first-year programming course, this finding is different in some significant ways. First, motivation and self-efficacy are *not* the same as course performance, the latter of which we were not able to compare to survey results. Secondly, most research on introductory programming has focused on courses taught through a computer science curriculum. While the concepts taught in IPC might not be much different from those courses, the fact that IPC is a required course for *all* engineering students at MAU might lead it have a different population of students than a typical introductory programming course. As an overarching goal of the redesign of this course is to help students without programming preparation learn how to program, we will continue to consider course design options that can continue to bridge the gap between students without preparation and

students with preparation, and continue to monitor changes to post-course survey responses after subsequent phases of the redesign.

Future Work

The IPC redesign is currently in Phase 2. This phase is introducing an array of course materials with the intent of facilitating student control over their own learning environment. This includes a course project, and interactive online materials. The final Phase of the course redesign will evaluate the possibility of dividing IPC into two courses, one for those with programming preparation, and one for those without. The final implementation of IPC will depend on results from assessments collected throughout these three phases.

References

[1] G. P. Wiggins and J. McTighe, *Understanding by design*. Alexandria, VA: Association for Supervision and Curriculum Development, 2005.

[2] V. Barr and C. Stephenson, "Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?," *Acm Inroads,* vol. 2, pp. 48-54, 2011.

[3] M. Smith. (2016). *Computer science for all*. Available: https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all

[4] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for K-12?," *Computers in Human Behavior,* vol. 41, pp. 51-61, 2014.

[5] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, *et al.*, "Scratch: programming for all," *Communications of the ACM,* vol. 52, pp. 60-67, 2009.

[6] A. Bandura, "Self-Efficacy," in *The Corsini Encyclopedia of Psychology*, ed Hoboken, NJ: John Wiley & Sons, Inc., 2010.

[7] B. J. Zimmerman, "Self-efficacy: An essential motive to learn," *Contemporary educational psychology,* vol. 25, pp. 82-91, 2000.

[8] A. Bandura and D. Cervone, "Self-evaluative and self-efficacy mechanisms governing the motivational effects of goal systems," *Journal of personality and social psychology,* vol. 45, p. 1017, 1983.

[9] B. D. Jones, "Motivating students to engage in learning: The MUSIC Model of Academic Motivation," *International Journal of Teaching and Learning in Higher Education,* vol. 21, pp. 272-285, 2009.

[10] A. M. Schmidt and R. P. DeShon, "Prior performance and goal progress as moderators of the relationship between self-efficacy and performance," *Human Performance,* vol. 22, pp. 191-203, 2009.

[11] R. W. Lent, H.-B. Sheu, D. Singley, J. A. Schmidt, L. C. Schmidt, and C. S. Gloster, "Longitudinal relations of self-efficacy to outcome expectations, interests, and major choice goals in engineering students," *Journal of Vocational Behavior,* vol. 73, pp. 328-335, 2008.

[12] R. W. Lent, S. D. Brown, and K. C. Larkin, "Self-efficacy in the prediction of academic performance and perceived career options," *Journal of counseling psychology,* vol. 33, p. 265, 1986.

[13] R. W. Lent, S. D. Brown, and K. C. Larkin, "Relation of self-efficacy expectations to academic achievement and persistence," *Journal of counseling psychology,* vol. 31, p. 356, 1984.

[14] B. D. Jones, C. Ruff, and M. C. Paretti, "The impact of engineering identification and stereotypes on undergraduate women's achievement and persistence in engineering," *Social Psychology of Education,* vol. 16, pp. 471-493, 2013.

[15] H. M. Matusovich, M. C. Paretti, B. D. Jones, and P. R. Brown, "How Problem-based Learning and Traditional Engineering Design Pedagogies Influence the Motivation of First-year Engineering Students," in *American Society for Engineering Education*, 2012.

[16] B. Jones, J. Osborne, M. Paretti, and H. Matusovich, "Relationships among students' perceptions of a first-year engineering design course and their identification with engineering, motivational beliefs, course effort,

and academic outcomes," in *annual meeting of the American Educational Research Association, Vancouver, Canada*, 2012.

[17] H. M. Matusovich, B. D. Jones, M. C. Paretti, J. P. Moore, and D. A. N. Hunter, "Motivating factors in problem-based learning: A student perspective on the role of the facilitator," in *American Society for Engineering Education*, 2011.

[18] B. D. Jones, M. C. Paretti, S. F. Hein, and T. W. Knott, "An Analysis of Motivation Constructs with First-Year Engineering Students: Relationships Among Expectancies, Values, Achievement, and Career Plans," *Journal of Engineering Education,* vol. 99, pp. 319-336, 2010.

[19] B. Jones, "User guide for assessing the components of the MUSIC Model of Academic Motivation," ed, 2016.

[20] P. Brown and H. Matusovich, "Unlocking Student Motivation: Development of an Engineering Motivation Survey " in *Americaon Society of Engineering Education Annual Conference*, Atlanta, GA, 2013.

[21] S. Bergin and R. Reilly, "Programming: factors that influence success," in *ACM SIGCSE Bulletin*, 2005, pp. 411-415.

Appendix A: Course Outcomes

Through this course, students will learn:

1. **An understanding of how computers work.** This will be demonstrated by an ability to:
   a. Describe the central processes behind all computing
   b. Describe the main strengths of computing
   c. Use bits to represent more complex abstract concepts
   d. Describe what a programming language is
   e. Describe what a computer program is

2. **An understanding of the fundamentals of programming using MATLAB.** This will be demonstrated by an ability to:
   a. Describe what variables are, in terms of bits and what they represent
   b. Set variables in MATLAB and give them values
   c. Describe the different types of information variables can represent in MATLAB
   d. Modify the value of variables with arithmetic and logical operators
   e. Assign values to variables via user input
   f. Write program that produces a desired output
   g. Use conditional statements to perform different operations depending on an input
   h. Describe the uses of conditional statements
   i. Use loops to repeat operations a desired amount of times
   j. Define the different types of loops and describe their uses
   k. Create functions that operate on a universal level
   l. Describe the advantages of user-defined functions
   m. Create more complex, modularized programs with multiple user-created functions
   n. Use some tools that are specific to the MATLAB programming interface

3. **An understanding of the broad usefulness of computer programming.** This will be demonstrated by an ability to:
   a. Use programs to solve engineering problems
   b. Describe some contemporary topics in computer programming and their applications
   c. Explore computer programming might be used in different engineering disciplines, including their own.