Congruency, Adaptivity, Modularity, and Personalization: Four Experiments in Teaching Introduction to Computing

David A. Joyner
Georgia Institute of Technology
Atlanta, Georgia, USA
david.joyner@gatech.edu

ABSTRACT
In January 2017, Georgia Tech launched a new online section of its CS1301: Introduction to Computing class. The course, offered both as a for-credit course to on-ground students and as an open MOOC, built on four unique design principles: congruency, adaptivity, modularity, and personalization. In this short paper, we describe the background of the course, the definitions of these design principles, and their application to the course design.

Author Keywords
Computing education, personalized learning, MOOCs.

ACM Classification Keywords
• Social and professional topics~CS1 • Applied computing~ Computer-managed instruction

INTRODUCTION
The internet is replete with places to learn computer science and computer programming. There are dozens of open textbooks, MOOCs, YouTube tutorials, interactive development environments, and more. A few months ago, Georgia Tech set about creating its own online Introduction to Computing course, and one of our first questions was: what is going to make this course different? The last thing the world needs is *another* online computing course, but are there needs that are not fulfilled by the courses that are currently out there?

In researching how to address this, we uncovered several places where a new course could distinguish itself. Some of these are largely administrative: like many MOOCs, this new online course is custom-built to take advantage of the options presented by the internet, but yet we are experimenting with offering it to on-ground residential students. It will also ultimately be offered as a publicly-accessible MOOC with Georgia Tech credit attached: any "students who successfully demonstrate mastery will earn a statement that may be recognized for credit if they later apply and are admitted to Georgia Tech" [2].

L@S 2017, April 20 - 21, 2017, Cambridge, MA, USA

ACM 978-1-4503-4450-0/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3051457.3054011

In addition to attaching credit to the course, however, we observed a number of experimental principles to leverage in the development of this course. Two of these, congruency and adaptivity, aim at creating a more complete pedagogical experience. These principles aim to inform a design for presenting content in multiple complementary mediums that adapt the learning experience to the student's current level of ability. The other two, modularity and personalization, inform a foundation for this course that preserves the potential to expand in new and innovative ways to encompass other programming languages and domains for application.

In this paper, we present the four principles that informed the design of this experimental Introduction to Computing course. It is important to emphasize that this course is very much a work in progress: it launched one week prior to this paper's submission deadline, and we are currently gathering enormous quantities of data to establish the usefulness and success of these principles and this course as a whole.

COURSE BACKGROUND
This Georgia Tech Introduction to Computing course is an online version of the school's foundational CS1301 course, which has no prerequisites for prior computing experience. Its designer and instructor (and this paper's author) is an award-winning instructor in Georgia Tech's online Master of Science in Computer Science (OMSCS) program, having taught and conducted research in it for two years [4]. The course features four primary technological components: the video course on edX, an adaptive textbook (authored by the instructor and built with McGraw-Hill Education), an automated evaluator for code (provided by Vocareum), and a digital proctoring service (Proctortrack, from Verificient).

The edX course is the central home of the course. The course is organized on the edX platform into nineteen chapters, each with an average of seven lessons. Each lesson is comprised of a handful of short videos (1 to 5 minutes), between which are interspersed interactive exercises (multiple choice and fill-in-the-blank). These exercises are graded for completion, and students have unlimited attempts to achieve the right answer; every wrong answer has dedicated feedback associated with it. Each chapter concludes with an additional page of suggested resources for further reading. The majority of chapters also

have an associated problem set, where students complete additional exercises but with a more limited number of chances per exercise (typically two). All of these are immediately and automatically evaluated.

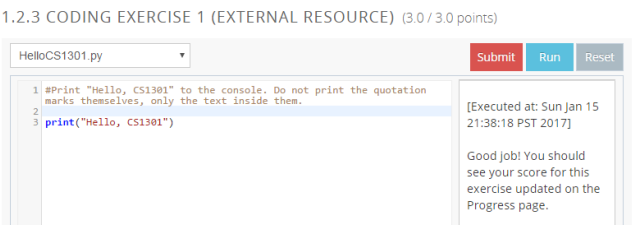

Figure 1: An example of a simple Vocareum coding widget. Here, the code window is on the left, and the window on the right shows the results of the student's submission. Students may also run the code on the left directly.

In addition to these multiple choice and fill-in-the-blank exercises, there are also programming exercises interspersed between the videos. These programming exercises all come with an embedded in-browser lightweight development environment, allowing students to write and run code directly in their browser. These exercises can be run against an instant auto-grader, which tests the code against a number of test cases and returns the result. These programming exercises are both interspersed in the chapters and included in the problem sets, with unlimited submissions in both locations. Additionally, each lesson concludes with a sandbox development environment featuring all the code shown in the lesson to allow students to easily jump in and experiment with the code featured in the videos. Exams are digitally proctored by Verificient's Proctortrack, but are comprised of the same kinds of exercises seen in the main course material, including multiple choice, fill-in-the-blank, and programming.

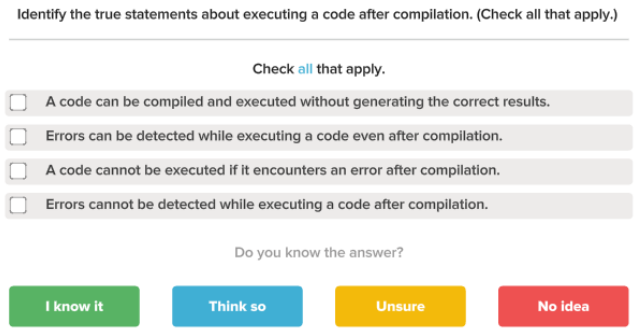

Figure 2: An example exercise from the adaptive textbook.

In parallel to this course is an adaptive textbook written by the course instructor and published on McGraw-Hill Education's SmartBook platform. As part of this SmartBook, McGraw-Hill Education has authored over a thousand exercises. Students complete these exercises as part of their completion of the edX course material. The next section contains significantly more information about the adaptive textbook, and an example exercise is shown in Figure 2.

Thus, a student's experience in the course is that each week, they complete one or two chapters and a problem set. In completing the chapters, they watch a series of short videos, most of which have exercises (multiple choice, fill-in-the-blank, or programming) interspersed at a rate of approximately an exercise per three minutes of video, and complete a series of exercises provided by the SmartBook. In completing the problem sets and exams, they complete similar sets of multiple choice, fill-in-the-blank, and programming exercises, with the added constraints of fewer attempts or digital proctoring.

CONGRUENCY

The first guiding principle behind the design of our Introduction to Computing course we dub "congruency". Congruency refers to a congruent structure between multiple presentations of the same course material. The principle of congruency comes from a lesson learned in Georgia Tech's OMSCS program. The program is built around several video-centric MOOCs, and students have repeatedly reported that while the production values and instruction in the videos are excellent, videos themselves are difficult to study. Searching, perusal, and rapid repetition are all unnatural interactions to have with a video compared to a textbook. To resolve this inadequacy, students have reflected on the value of transcripts, and a couple classes have gone so far as to share the course scripts or transcripts in a more textbook-like format that students can use to more easily seek the target material.

In many ways, this observation is consistent with a residential experience. Instructors assign class readings that overlap with lecture material because it allows the same material to be presented in two different ways. However, oftentimes mentally mapping the lecture material to the reading material requires expert-level knowledge in the first place.

This is where our principle of congruency comes in. As noted, the course is made of two primary sources of material: a video-based course on edX and an adaptive textbook on McGraw-Hill's SmartBook platform. These two sources of material, however, are congruent in their content, structure, and examples. They organize the content in the same way, use the same examples, and show the same visuals. Each chapter of the course contains a dedicated widget to launch the corresponding chapter of the SmartBook.

The goal is to facilitate easy alteration in the medium from which students choose to consume content. We hypothesize that students will generally choose to initially consume the material from the video-based course, but will use the textbook to recap the material later, take a deeper dive into some of the course material, and more slowly move through material they find confusing.

Complete data is recorded on students' interaction with both the video course and the adaptive textbook. This data

will be used to create profiles of students' interaction patterns and connect those patterns with learning outcomes.

ADAPTIVITY

Adaptivity is not a particularly new idea in computer-assisted instruction; the general area of intelligent tutoring systems has built on computer-aided adaptivity for decades [e.g. 1, 6], and efforts are already under way to extend such features to online education [e.g. 3]. However, most similar efforts focus on adaptivity specifically within a practice environment with dedicated feedback. This experiment in teaching Introduction to Computing aims to instead integrate adaptivity into the instructional process.

This is achieved in two ways. First, as noted, the course and adaptive textbook are tightly integrated, and the adaptivity in the textbook comes from a collection of several dozen exercises for each chapter. These exercises are each tied to a learning objective present in the textbook, and as students complete these exercises, the platform constructs a model of students' mastery of those learning objectives. When students answer incorrectly to a particular exercise, they receive feedback from the textbook on the correct answer; however, they will then later be re-tested on the same learning objective using a different question to ensure they are developing their understanding of the material rather than simply recalling answers they have already seen.

Students' experiences within the textbook then change based on their current level of mastery of the objectives as communicated via the exercises. If a student continues to struggle with a certain learning objective's exercises, the textbook directs the student to the area of the book that covers that material. The congruency described previously also allows students to then jump to the identical corresponding area in the course videos, which are also launched from within the textbook. Additionally, whenever a student peruses the textbook, the adaptive platform applies a visualization on top of the text calling students' attention to the areas in which they have already demonstrated mastery in the exercises, as well as the areas in which they have struggled or not yet demonstrated mastery. In this way, the textbook experience adapts to students' current level of ability.

Similarly, the exercises integrated into the edX course facilitate some adaptivity as well. Each exercise is constructed with dedicated feedback on anticipated wrong answers. While this boils down to a straightforward mapping between answers and feedback in multiple-choice and fill-in-the-blank questions, the programming exercises allow additional adaptivity. Each programming exercise is itself evaluated by a Python script that can examine both the output of students' code and the code itself, allowing a complex tutoring system to be built that evaluates code style, efficiency, and function, along with providing dedicated feedback based on anticipated incorrect answers. In its initial incarnation, the feedback supplied by this system is largely implicit (such as providing students desired output to compare to their code's actual output), but as a library of past student answers and mistakes accrues, detailed feedback will be developed based on the most common patterns of errors.

MODULARITY

The third guiding principle of the course's design is potentially controversial. Modularity in this context refers to a modularity between three general topic areas in computer science: foundational concepts, language fluency, and domain applications. Each video of the course falls into one of the three categories. Foundational lessons do not use any particular language's code; they focus on more abstract concepts. Language lessons then take those concepts and concretize them in code with actual syntax and execution. Domain lessons then take those principles (and sometimes that language) and apply them to a particular application domain, like computer graphics, data science, or robotics.

There have been unsuccessful efforts in the past that attempted to teach foundational concepts separate from instructing their application in a particular language. These have been unsuccessful due to the observation that understanding of core concepts is tightly tied initially to the syntax in which they are written; higher-level understanding comes with practice *with* that syntax, not from learning the concept prior to that syntax. We hypothesize, however, that our effort will be more successful because of a specific affordance of the online medium: whereas some efforts have split foundational concepts and language fluency into long, entirely different lectures, we instead rapidly switch between them. Five minutes of foundational material will be followed by five minutes of implementation of those concepts in a particular language before switching back to foundational concepts. We posit that this mirrors the way the subject matter is actually taught, and the online medium simply affords us the ability to concretely but rapidly switch back and forth between areas.

There are two goals of this modularity. First, it aims to equip students with an understanding of the fundamentals of computer science in addition to fluency with a particular programming language. Second, while we believe that this modularity will present a valuable way of learning computer science on its own, modularity is also a means to an end. Specifically, the plans for the course's personalization are derived from this modularity.

PERSONALIZATION

In its initial state, the course teaches Introduction to Computing in Python with computer graphics as its domain of application. However, the modular design of the course is with a strong eye toward individualization. Modularity in this sense is intended to allow for easy substitutions within the different areas of the course such that an experience may be constructed that is personalized to the learner's own interests.

In the near-term, the major application of this is expected to be in the domain material. The initial deployment of the course emphasizes computer graphics as its domain, but the advantage of the online environment is its potential to create an experience that allows students to choose their own learning path. Toward this end, additional domain modules are planned focusing on other topics, including robotics, data science, and artificial intelligence. Given that the class is often taken by students majoring in topics like engineering, science, business, and arts, additional modules are planned that focus on those topics. The hope is that while the course launches as an "Introduction to Computing", these domain options will allow for dynamically personalized courses like "Introduction to Computing for Accountants" and "Introduction to Computing for Musicians."

A second phase of personalization comes from the modularity of the language component. By separating out the foundational concepts from the language component of the course, the entire course could be redeployed in a different language by replacing only ~50% of the content rather than 100%, and much of that content demands only a syntax translation rather than a wholesale rewrite. Comparable Introduction to Computing classes are often taught in Java, Matlab, and C++, and ongoing trends suggest there may arise a demand for Introduction to Computing in Swift, Ruby, or JavaScript. Domain material could complement those as well, especially with popular JavaScript frameworks. Thus, with far less work than creating an all-new course, this Introduction to Computing could become "Introduction to Computing for Engineers in Matlab" or "Introduction to Computing for Graphic Designers in Swift."

Finally, an ideal third phase of personalization may come from the options to select instructors and spoken languages. To increase inclusivity, this drive for personalization may allow students to select an instructor based on the desired gender and race from which they would like to learn. This approach will allow us to showcase the diversity of individuals finding success in the computing field, thus letting students select an instructor who will most personally resonate with them [5]. Similarly, by translating the course into other languages, we hope to extend its availability to students around the world.

CONCLUSION: TO BE CONTINUED…

This Introduction to Computing course is an experiment in a number of different ways. First, the four principles outlined here that have guided the structure of the course are themselves experiments. We may discover that students do not leverage the congruency between presentation styles at all, or that the modularity confuses students more than it supports them.

The course represents an experiment in other ways as well. At the most general level, it is an experiment to see if an online course can succeed for a residential audience. It is similarly an experiment to see if the principles and expertise cultivated in a Master's program with at-a-distance students translate to an undergraduate program with residential students. Other experiments include whether or not access to a live development environment during test-taking enhances learning outcomes and whether or not an online course draws a different type of student compared to residential classes even from the same student body. Every element of this course is set up to learn, improve, and iterate through experiments like these.

ACKNOWLEDGMENTS

We are grateful to our partners in developing this course: Georgia Tech's College of Computing (especially but not limited to Zvi Galil, Charles Isbell, Melinda McDaniel, and Bill Leahy), Georgia Tech's Center for 21st Century Universities (especially but not limited to Pam Buffington, Jo Keith, Rob Kadel, Amanda Madden, and Rich DeMillo), Georgia Tech Professional Education (especially but not limited to Yakut Gazi, Shabana Figueroa, Brian Wilson, Stephen Murphy, Brian Armstrong, Aqueelah Sabir, and Nelson Baker), McGraw-Hill Education (especially but not limited to Tom Hinkley, Cal Alford, Jenny Bartell, Amber Cortez, Stephanie Wilson, and David Levin), edX, and Vocareum. We are also grateful to the teaching assistants authoring some content and helping run the first offering of the class: Marguerite Murrell, Joshua Diaddigo, and Jackie Elliott, Rachel Golding, and Christine Feng.

REFERENCES

1. Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*, *228*(4698), 456-462.
2. Georgia Tech. (2016). Taking Undergraduate Computer Science Online. Retrieved from http://www.cc.gatech.edu/news/583367/taking-undergraduate-computer-science-online
3. Heffernan, N. T., & Heffernan, C. L. (2014). The ASSISTments ecosystem: building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, *24*(4), 470-497.
4. Joyner, D. A., Goel, A. K., & Isbell, C. (2016, April). The Unexpected Pedagogical Benefits of Making Higher Education Accessible. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale* (pp. 117-120). ACM.
5. Kizilcec, R. F., Saltarelli, A. J., Reich, J., & Cohen, G. L. (2017). Closing global achievement gaps in MOOCs. *Science*, *355*(6322), 251-252.
6. VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, *46*(4), 197-221.