SPLICE: Self-Paced Learning in an Inverted Classroom Environment

Matt Boutell and Curt Clifton
Rose-Hulman Institute of Technology
5500 Wabash Ave.
Terre Haute, IN 47803-3999
812.877.8534

{boutell, clifton}@rose-hulman.edu

ABSTRACT

Learning to program is hard for many students. Practice with an expert coach is key to overcoming this challenge. However, finding time for this is an issue because presenting concepts, showing examples, and modeling problem solving reduce the time available for mentored practice. Pace is also an issue because some students arrive with confidence and prior experience and are thus bored, while other students labor and become overwhelmed.

To address these problems in CS1, we created on-line videos for a C programming unit to present concepts, show examples, and model problem solving. As a result, our students spend every class session entirely in active learning activities with expert coaching, receive more individual attention, and set their own pace.

1. SIGNIFICANCE AND RELEVANCE

Anyone who has taught introductory computer programming courses understands that *learning to program is hard for many students*. Most students readily grasp the basic pieces, but they fall down in their logical thinking. They struggle to compose the basic pieces to create software that solves a real problem. We often hear some students say, "I understand all the concepts in class, I just can't program." Their exam performance supports that claim: these students can answer conceptual questions, but have difficulty actually solving problems by programming.

But solving problems by programming is the essential skill. Even in our introductory courses, we're focused on helping students achieve competence at the *application* level in Bloom's taxonomy. Reading through the course outcomes we see words like *design*, *implement*, *test*, *debug*, *demonstrate*, and *solve*. To achieve this level of learning, students must practice.

Before students can practice, though, they need to learn basic concepts (like syntax and control structures); get to know their tools (like integrated development environments); and watch an expert model the problem solving process. Then they can move into structured practice, where they fill in pieces of the puzzle under the guidance of an instructor and with appropriate scaffolding. At this stage, students are developing comfort with the basic concepts and tools, while the instructor coaches them. Although students could do this on their own, many spend a significant part of their work time on incidental challenges, like syntax or obscure error messages, rather than the core concepts. A coach can help the students move past the incidental challenges and focus on the essential challenge of logical thinking.

As students build confidence, we gradually pull away the initial scaffolding and ask them to solve more realistic problems. At this stage, expert coaching and feedback are even more important. An attentive instructor helps students continue to develop their skills and confidence. Ultimately, students are able to work in teams, independent of the instructor, on larger projects.

This progression of activities, from understanding basic concepts through independent problem solving, is why we adopted a studio format for our introductory courses. In a single class session, students learn a new concept, experiment with it, and apply it to a real problem, all with expert coaching at hand. While this format has been effective, in practice, we still find two significant problems: time and pace.

Time: We've found that we spend a majority of our class time, even in a studio format, on the earliest stages in our progression of activities. Although textbooks can describe the basic concepts, they are abstract, so we spend some class time reiterating that content. A textbook cannot provide examples of the dynamic problem solving process, so we spend class time modeling that. Students report that they appreciate these *live coding* examples, where an instructor programs in front of the class and thinks aloud about his or her problem solving process. Unfortunately, live coding is not very interactive and consumes substantial class time—time that would be better spent providing expert coaching.

Pace: The other problem we face is finding the right pace for our introductory courses. It seems like regardless of the pace chosen, midterm surveys and course evaluations show that some students are bored while others are overwhelmed. While programming takes time for most novices to learn, some students need even more time. At the opposite end of the spectrum, other students, either due to prior experience or great aptitude, quickly become bored because the pace is too slow for them. This compounds the problems of the slower students, who become intimidated not just by the material, but also by their peers. This "erosion in confidence" is particularly pronounced for women [1, ch. 5]. Furthermore, because the distribution of students tends to be fairly uniform between the two extremes, using any lock-step approach, regardless of pace, fails to meet the needs of many of our students.

To address the complementary problems of time and pace in CS1, we replaced lectures, examples, and live coding in a C programming unit with on-line videos. This maximizes the amount of class time that students spend in mentored practice and allows students to control the pace at which they digest the material. The videos enable self-paced learning in an inverted classroom environment, or *SPLICE*.

The SPLICE videos fall into two categories. *Concept videos* introduce new concepts with short screencasts combining slides and examples. These videos supplement and reinforce the concepts presented in the course textbooks. *Live coding* videos show an expert who models logical thinking by solving sample programming problems while thinking aloud. These videos

present a thought process that cannot be readily conveyed by a traditional textbook.

Students watch these videos to prepare for class. In the process, they complete active learning exercises, such as answering quiz questions and writing small code snippets. This helps them engage with the material. To give students immediate feedback, a later portion of each video gives solutions to the coding questions.

By using these videos, students arrive in class prepared to practice the ideas to which they've already been exposed. We give them an assignment over the material and they get right to work. The instructor circulates, observing their work and offering appropriate assistance. In class, students' attention is focused on active learning, while the instructor's attention is focused on coaching his or her students.

Using videos for instruction allows students who would have struggled with concepts will be able to rewind and watch tricky segments a second time. These students also benefit from a greater share of the instructor's time—time that can be spent identifying the particular and individual sources of a student's confusion. Furthermore, students who would have found the pace too slow are able to work quickly through material that they already know and delve into more interesting problems.

A significant benefit of video lectures is their broad availability over space and time. First, they are available to students learning on their own or at other institutions worldwide via YouTube.[1] Second, any student who has used the videos can reuse them for reference when taking courses that expect familiarity with C, like operating systems or computer networks.

The inverted classroom environment [2], in which lecture is moved before class to make room for other in-class activities, is not a new idea. Kaner and Fiedler [3] and Day and Foley [4] used video lectures to invert their upper-level software courses. These videos were merely a "talking head" in the corner of PowerPoint slides, with no active learning component for students while watching the videos. Furthermore, class time wasn't used for hands-on application of the ideas, but for further discussion of the concepts. Gannod, *et al*. [5] used video podcasts to deliver full lectures successfully to students in their introductory computer science course, but their videos are not freely available.

In general, audio and video instruction to teach computer programming is also not new, though we have not found anyone who provides the same mix of concept coverage and sample problem solving. Georgia State offers an Intro to Programming in C# video course (as part of Apple's *iTunes U* program), but it is incomplete [6]. Trinity College (Dublin) offers Intro to Computer Programming in C++ on iTunes U, but only some sessions are available. The course is also specific to Mac OS X. UC Berkley's Computer Science 61C contains five lectures on introductory C programming, but this course is the third in their CS sequence, so presumes two full terms of programming experience. The Berkley course is on iTunes U but is audio-only. MIT's Open Courseware includes EECS 6.00 Introduction to Computer Science and Programming in Python [7]. This is a very different curriculum than ours, taking a breadth-first approach. It is also not a screencast, but a professor at a blackboard teaching concepts. Stanford offers full video lectures for a three-course introductory software development sequence as part of the Stanford Engineering Everywhere initiative [8]. This sequence begins with Java. C is not covered until the third course. At the Naval Academy, Carlisle assigns 5-minute YouTube clips to his programming students, but the course is in Java [9]. A variety of commercial training videos on C programming are also available on-line, but they are all incomplete, inaccessible, or inappropriate for our course sequence and tools [10-12].

[1] Our videos will be posted to YouTube following our initial assessment stage.

2. POSTER CONTENT

In this poster, we will summarize the motivation for using self-paced video instruction and expert coaching. We will provide screenshots of both types of videos and examples of the assessments students complete while watching the videos. We will provide details of producing the videos, such as instructional design, the effort required, and the environment needed. We will include preliminary assessment of the approach, comparing feedback and grades from three sections of CS1 taught using the videos to that from three sections taught via traditional lecture. We will include future plans to add more self-pacing, to expand this approach to other topics, and to perform a detailed, longitudinal assessment. We will also demonstrate each of the videos on a laptop.

3. ACKNOWLEDGMENTS

This work was funded by a Rose-Hulman Summer Professional Development grant.

4. REFERENCES

[1] Margolis, J. and Fisher, A. 2002. *Unlocking the Clubhouse: Women in Computing*. MIT Press.

[2] Lage, M., Platt, G., and Treglia, M. 2000. Inverting the classroom: A gateway to creating an inclusive learning environment. *J. Econ. Educ.* 31, 1 (Winter, 2000), 30-43.

[3] Kaner, C. and Fiedler, R. 2005. Blended learning: A software testing course makeover. *11th Sloan-C Int. Conf. on Asynchronous Learning Networks*, Orlando, FL, Nov. 2005.

[4] Day, J. and Foley, J. 2006. Evaluating a web lecture intervention in a human-computer interaction course. *IEEE T. Educ.* 49, 4 (Nov. 2006), 420-431.

[5] Gannod, G. C., Burge, J. E., and Helmick, M. T. 2008. Using the inverted classroom to teach software engineering. *30th Proc. Int. Conf. Software Eng.*, Leipzig, Germany, May 2008, 777-786.

[6] http://www.apple.com/education/itunes-u/ (accessed 4/7/2010).

[7] http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/index.htm (accessed 4/7/2010).

[8] http://see.stanford.edu (accessed 4/8/2010).

[9] Carlisle, M., 2010. Using YouTube to enhance student class preparation in an introductory Java course. *SIGCSE Bull.*, Milwaukee, WI, Mar. 2010.

[10] http://www.computer-training-software.com/c-programming.htm (accessed 4/5/2010).

[11] http://www.softwaretrainingtutorials.com/c-programming.php (accessed 4/5/2010).

[12] http://showmedo.com/videotutorials/series?name=MjNtBGUsy (accessed 4/5/2010).