Abstract

Faculty member and researchers will present the results of a pilot flipped classroom Introduction to Programming course. The course leverages the Swift programming language and introduces iOS app development concepts alongside the programming fundamentals. Lectures are delivered through a series of 70 plus videos, which students consume outside the class, following along while developing a series of apps. Videos are wrapped in learning scaffolding including a summary reference and immediate-feedback online quiz. Students spend class time with Mac laptops in light review, but mostly working collaboratively on problem exercises. Course content is being made available to other faculty and it is hoped that this approach and content affords a light-lifting method for faculty to offer a current, high-demand learning opportunity. Faculty and student experiences will be shared, along with challenges and areas of improvement.

Keywords


Introduction

Undergraduate technology-focused enrollments have been rising for several consecutive years (Steed, 2013).

The continued high demand for technical careers has helped fuel student interest, as has a heightened interest in collegiate entrepreneurship. Indeed, the advent of the app store, cloud computing, and open-source software, combine to create an environment where the cost associated with developing a technically-oriented product or business has fallen within the range of many undergraduates.

Despite high initial interest, there are many roadblocks in the path of technical skills acquisition. Curricula heavy in programming also experience among the highest student attrition rates. Many would-be student entrepreneurs struggle with the learning curve required to gain the skill to implement their vision. Student skill levels vary and many new students can easily feel left behind in a classroom where others seem to arrive with a skillset that gives them an advantage.

Faculty also struggle. In information systems, technical skill acquisition is often a challenge for faculty. Many IS faculty have research agenda that don't directly benefit from the prep and delivery associated with a programming course. Modern development tools and programming languages are in a regular
cycle of iterative change and improvement, further taxing the ability of time-strapped instructors to develop and refine technical courses. Programming classes can also offer an increased need for faculty-student out-of-classroom interaction, as students struggle with challenging problem sets, trip up as they encounter syntax challenges common among new programmers, and more.

A New Approach

The authors have been involved in various aspects of planning, executing, and researching the results of a flipped classroom approach to a students' first programming course. The flipped classroom pedagogical method employs asynchronous multimedia lectures and exercises that students consume outside of traditional course lecture time, while in-class work is made up largely of additional exercises and group problem solving activities (Roteller and Cain, 2016).

There are a number of factors that make the flipped classroom a particularly attractive area to explore as the pedagogical organization of a computer programming course (for a survey of related research, see Bishop and Verleger, 2013). There is a long history of studies have shown that video-delivered lectures are often at least as effective as in-person lectures at conveying basic information (Cohen et al, 1981; Zhang et al., 2006), and these benefits may be even greater for students learning programming. Since students learn at different rates and arrive with disparate skillsets and experiences, students who struggle can pause, go back, and re-watch video lectures. Video lectures delivered on computer can be combined with student hands-on work, where learners follow along with the lecture, performing the same tasks and techniques as they are introduced in the video, rather than taking notes and applying their knowledge outside of class after some delay.

Dense technical topics can be especially challenging for lecture-based course delivery, as well. Many technical topics build on earlier learning, so if a student misses or fails to grasp even a portion of a lecture, they may struggle for the rest of the course. However, video lectures allow a student to never miss a lecture, and to regularly review a topic in its entirety in a way unavailable in a traditional classroom (Milman, 2012; Herreid and Schiller, 2013).

The acquisition of communication and other so-called "soft skills" remains in high demand in technical fields, although many new employees arrive without adequate preparation for interpersonal team and client work (Andrews and Higson, 2010). Group problem solving in class may have additional benefits for technical education. Having students discuss a problem and teach each other should enhance communication skills vital for technical team members.

Material developed for and shared with others for use in the flipped classroom model may also involve far less prep and student support time than a conventional class, freeing faculty to focus on higher-quality student interactions research, or other professional demands. Since lectures have already been developed and are delivered through the platform, faculty don't have the extensive prep involved in learning, developing, and testing new programming concepts. A flipped class offering that provides in-class exercises, solutions, and an awareness of common student struggles should also minimize the time that faculty are "on", moving them from delivering full classes of one-to-many lectures, to a new role where they are facilitating and supporting group interaction. Faculty may find that they can rely on student teaching assistants to help support in-class collaboration and out-of-class student questions and struggles, further lightening the load as compared to a more traditional programming course where the faculty is the primary content deliverer.

Implementation

The development of the current course came about largely from student demand. The University's current programming offerings are very favorably received. Instructors regularly receive high student evaluations and interest in the information systems major has grown and remains strong. However, students have found that current introductory programming offerings are light on the most current technologies, especially in mobile app development. Legacy tools such as Visual Basic, can introduce a degree of creativity, but lack the currency and appeal of mobile development. A focus on text-delivered languages like Python, can be less motivating than a course using tools that are visually rich and have an engaging user interface.
Tool Choice

We set out to try to redesign a student's first collegiate course in programming so that they could learn a programming language while simultaneously developing visually-rich applications. The environment chosen was the Swift programming language focusing on iOS app development. Originally developed by Apple, Swift is now open-sourced and available on many platforms. iOS development is accessible using Apple's free Xcode development software and iOS simulator tools. While students need access to a Mac, they don't require their own mobile device. The dominance of Apple products among the student body and the relative sophistication of Apple's tool set, along with Swift's emergence displacing Objective C as the language-of-choice for professional Mac and iOS development, led to this selection. Xcode also offers an interpreter environment, known as the Playground (not to be confused with a similar iPad Playgrounds app). The Playground makes it easy to experiment with coding concepts, then move work to the quickly-compiled project environment to run on the iOS simulator. Student programming work from following along with videos is submitted via GitHub and graded each week by course TAs.

Concept Introduction Alongside UI Development Skills

The course is designed around the development of a series of apps. Students are assigned roughly 2 ½ hours of video lectures each week, with the understanding that it may take two to three times that to follow along and complete hands-on work. The first two weeks of video lectures introduce students to fundamental topics such as constants, variables, arrays, randomization, and functions. Students build a personal app, "You Are Awesome!" which shows random positive messages, encouraging photos, and cheering sounds. Subsequent apps focus on introducing standard programming topics alongside increasingly sophisticated iOS app design. The introduction of arrays, for example, provides an opportunity for the table view interface element to be introduced (a scrollable list of cells) in a To Do List app. A metric conversion app uses the Swift concept of closures (code that can be stored and passed like a variable) to house conversion formula accessed through labels in a scrolling wheel. Students learn the fundamentals of APIs and data parsing by using the relatively easy-to-implement Google Places library to auto-fill place locations and gather latitude and longitude data that is passed to the Dark Sky weather site and returned for a personal weather app.
You are Awesome – Who needs "Hello World!" when your first app shows a series of random images and sounds proclaiming your awesomeness!

Bip the Guy – Add photos of your favorite nemesis or take photos with the camera, then tap to "punch". Complete with punch sounds and spring in/out animation for that satisfying "Bip".

To Do List – Get organized while you learn how to add, update, and permanently save list data. You'll learn lots of skills found in many professional apps.

ConvertIt – Creating a metric / imperial units conversion app is a great way to learn new user interface elements, and to expand knowledge of "Swifty" programming concepts.

WeatherGift – An app you'll want to share with loved ones. Add your own photos that will show up inside this real-time weather app, varying photos depending on the current weather. Learn to get and
parse real weather data and other information over the web, get the user's current location and use GooglePlaces to add new locations and work with different times and dates.

Table 1. A Sampling of Apps Developed in the Course

Learning Scaffolding and Reinforcement

Each video is wrapped in learning scaffolding. Apps are introduced as "Chapters" which may run two chapters in a week, to several weeks to a chapter (a completed app). Chapters start with a demonstration video showing students what their completed app will look like, and will introduce new terms associated with the interface elements to be implemented. Each chapter section includes a video focused on a clear topic or set of topics to move the app toward completion. A reference section follows each video, often including images and animated .gifs demonstrating techniques, and providing a summary list of steps performed and why they were performed. The reference sections are meant to function as a go-to step-by-step reference for the steps needed to implement more complex topics.

The reference section is followed by a Key Takeaways bullet list, and an online quiz (multiple choice, true false) offering students immediate feedback on whether or not they've understood concepts introduced.

Exercises following each section are largely meant to be done in class, however many of the sections include "challenge problems" that briefly ask students to apply a newly acquired skill for reinforcement and to confirm understanding. Students aren't discouraged from completing all exercises outside of class, but in the course's current form, this would involve a significantly larger amount of work, so students are told in advance if there are specific homework problems and to otherwise prepare to complete the rest in class.

Several faculty at the University had initiated flipped classroom experiments of their own. Several science classes, and an accounting class have used this model. Through initial discussions with pioneering faculty and the University's Center for Teaching Excellence, faculty understood that students prefer at least a brief reinforcement of course concepts before diving into problems. This is usually no more than 15-20 mins reviewing slides and small coding examples demonstrated in the week's videos.

Problem exercises are then put up on a video projector, and students are also referred to the section of the online "book" that contains the problem. Students are encouraged to work together and to ask the neighbors or neighboring teams for help and hints. Students are also encouraged to raise hands high if they struggle or need assistance, with a member of the TA team or the instructor stopping by to offer hints and help. Faculty wanders the room to get a sense of student learning. This is a unique opportunity to overhear in real time what students are grasping and what they are struggling with. It helps point out students who consistently struggle or who are stars, and it highlights if some material has been difficult for all to grasp, perhaps requiring additional instruction. Problems are always "solved" on the instructor's Mac after an adequate period of group work, but with regular call out to students asking their logic in constructing a solution (and often resulting in multiple interesting solutions for a single problem). Many of the in-class problems involve building short "fun" apps (a Magic 8 Ball, the Harry Potter Sorting Hat), or include pop culture references alongside computing topics (e.g. using an array of Strings consisting of popstar names, write a function for Taylor Swift to return any array passed with Kanye West removed from the guest list). Energy in completing these tasks is high, with smiles alongside head-scratching frustration as students push the limits of their coding skillset. Some samples of the pedagogical material are included in the appendix, along with a link to a sample video lesson.

Course Execution and Organization

All course material is offered online. The "text" of videos (recorded and served from YouTube and embedded in web pages for each of the text's sections) and supplemental course material were developed in a content creation and delivery software product from the publisher Flat World Knowledge. Online quizzes were initially offered using the online ClassMarker platform, but will eventually be migrated into the Flat World tool. Students use Apple's Xcode IDE for their work, and load completed work to personal GitHub accounts (GitHub access is built into Xcode). Course TAs grade weekly GitHub submissions and gather automatically-tallied online quiz results.

The course also has regular short in-class written quizzes. Two hands-on mid-terms (one focusing largely on Swift programming fundamentals, the other focusing on iOS app development fundamentals), are
completed in class and uploaded to the school's Canvas course management system. Mid-terms are challenging enough that most students use the entire class time. Faculty and TAs can easily scan the room to ensure that students aren’t cheating or accessing materials not allowed for the test. Hands-on exam work is much more representative of problem solving, given the ability of modern IDEs to catch and correct simple syntax errors, freeing coders to focus on their logic.

A final exam will combine Swift and iOS concepts in a 2 ½ hour evaluation, similar to the two mid-terms. Students will also complete an independent project, developing an original app of their choice. The last class will be student app showcase, inviting in alumni and area employers to network with our students, providing additional mentoring contacts, motivation, and hopefully even internship opportunities.

The course also leverages a class wiki to solicit student ideas for possibly inclusion as exam questions. Students are required to submit questions and answers. While faculty commit to offering no more than 70 percent of exam questions “inspired by” student input, the wiki-submitted questions provide a useful tool to expose a tool for student self-assessment. If she can complete student submitted questions, this is likely an indicator of exam success. Struggling with suggest special areas to review and focus further study. Faculty often struggle with issues of grade inflation, and while a serious issue, the concept is often poorly framed. Faculty are not tasked with producing a bell curve of results, they should be focusing on setting a high bar for student excellence, exposing that bar, and providing resources to help as many students as possible exceed that bar. The wiki is a most effective tool in helping accomplish this.

Finally, the course has been using social media with a class hashtag. Faculty regularly tweet photos of students at work, and teasers of assignments being worked on. Several firms have expressed interest in outcomes for potential intern hires, and former students have often shared envious comments on wishing they could also take the course.

**Results**

The work is part of ongoing, emergent research and as of this writing the both course mid-terms are complete and students have built four full apps in the course. All online videos and course content have been created, as well as several hours of optional lessons for students who wish to expand their skills with additional techniques to use in final projects. The forthcoming conference presentation will report findings, anecdotal comments, photos and video of student work (including the final student app showcase), survey, post-survey focus group comments, and course feedback results.

Students have significantly praised the approach. After the first week, students were spotted on campus showing apps to their friends. Some of the apps include components for personal photos that they can share with loved ones as gifts. Work from one of the pilot course students was awarded the GE Prize for "Best Hack for the Social Good" at the university's "Hack the Heights" 24 hour hackaton, with the recipient citing "I used much of what we learned in our Swift class to develop the user interface, like setting up the map, an additional ViewController, a UIPickerView, and segues." And many students have spoken outside of class with faculty and TAs about continuing their technical studies or their interest in entrepreneurship. A course admitting 50 percent more students is already oversubscribed for the Fall term.

**Challenges**

One of the challenges with video-driven learning is ensuring that students are acquiring skills and not simply parroting keystrokes and mouse clicks. It is hoped that the online quizzes and in-class exercises will minimize this risk, but initial feedback suggests faculty are still fine-tuning the right balance between video learning and immediate reinforcement through exercises.

The course is now being offered only to those students who have never taken a collegiate computing course. While many students have done well, some have struggled. Course grading is being modified as needed to try to encourage students to advance and not be left behind. Initial strategies considered are retaking mid-terms at a reduced total point percentage, or completing similar but different mid-terms to encourage additional effort toward mastery.

Access to hardware is also an issue for some. The University has fortunately offered loaner Mac laptops that were requisitioned when older machines were retired. Still, if students experience hardware
problems for tools required for course use, this can present challenges and faculty need to think about potential hardware failure issues during exams or problem work, and how they would solve these.

Future Study

We intend to assess student self-perception of themselves as innovators and change in sentiment of their intent to pursue technical careers and further technical courses. We will specifically draw on the work of (Marsh & Roche: 1997; Centra: 1979; and Feldman: 1989), which informs us as to the importance of student perceptions of how this innovative learning environment affects them in two specific aspects. Firstly, students' perceptions of themselves as innovators and entrepreneurs. Secondly, how the innovative learning environment affects students' self-perception. This will be explored by using a survey with 12 close-ended and two open-ended questions. The goal of this analysis is to identify areas of inquiry for a more rigorous pre and post survey of students that would allow changes in perception to be measured with greater effectiveness. The survey will include some background information of students (years of study, major, GPA) and explore whether there is any significant association between background variables and student perceptions that could give some insight regarding the optimization of course methodologies/materials.

We also hope to longitudinally study students and compare them to others, testing whether the class has led to increased student innovation, increased internship opportunity, an improvement of self-confidence, and other factors.

REFERENCES


Appenndix

Class Exercise: In the West African nation of Ghana it is common for people to have a 'day name', that represents the day of the week when they were born. Create a new Xcode project "GhanaName" that contains a picker wheel with the days of the week (Sunday through Saturday). When the user selects a day, labels below the picker should show the appropriate male and female Ghanaian "Day Names" for that day. The Wikipedia article titled "Ghanaian Name" should show you the various options by day and gender. What would your Ghanaian Day Name be?

Figure 1: Sample of online text / learning scaffolding

Figure 2: Sample of in-class exercises
1. **USE YOUR STRUCT ARRAY IN A FUNCTION TO FIGHT THE POWER:**
You've heard reports that women, on average, earn 77 cents for every $1 earned by similarly qualified males. This is outrageous! You realize that if this is happening to your firm it's a huge problem – you're competing for talent, and a workplace environment that isn't perceived as equitable with competitive compensation won't be able to appeal to the widest possible talent pool. [10 pts.]

Write a function `genderSalaryAverages` that takes the `employees` array of struct `Employee` that you defined above, and internally calculates `maleAvg`: an average salary for male employees and `femaleAvg`: an average salary for female employees. The function should then examine the results and print "Men earn an average of: `maleAvg`, Women earn an average of: `femaleAvg"", substituting, of course, the correct numeric averages for both `maleAvg` and `femaleAvg`. [4 pts.]

Also examine the results and print guidance for your firm:

- If the difference between male and female average salaries has men earning an average of $5000 or more, print "Women are paid notably less than men. We've got to change this. It hurts our recruiting and is a lawsuit waiting to happen"
- If the difference between male and female average salaries has women earning an average of $5000 or more than men, print "Women are paid notably more. That's pretty progressive, but we should really have equity among all employees. Fix this."
- If the difference between male and female average salaries is less than 5000 print: "We don't have a notable difference in male vs. female salaries. Good work!"

2. **PROJECT (BUILD AN APP) [10 pts.]:** Write an app: `sortingHat` that accepts a student's name from a textfield and, when a button labeled "Sort!" is pressed, will display in a label: "name is in houseName" where name is the student's name and houseName is one of four randomly chosen Hogwarts houses: Gryffindor, Hufflepuff, Ravenclaw, or Slytherin.

- Set the App's background color to dark gray.
- Set the label to display in what is roughly equivalent to the main house's color: red for Gryffindor, Yellow for Hufflepuff, Blue for Ravenclaw, and Green for Slytherin.

The app should look roughly like the one below:

Figure 3 Sample Exam questions
7.20 Working with Dates, Times, Date Formats, and Time Zones

**Learning Objectives**

- Learn about various computer date/time formats, and how to use Apple’s TimeInterval, Date, DateFormatter, and other data structures and methods to manage and properly format dates and times, accounting for time zones.

**Video Lesson**

![WeatherGift 15 Dates](image)

**Reference**

**Date Formats and Format Conversion**

Computers typically express dates and times as numeric values that aren’t easily interpreted by humans. Some of the formats you

Link to a sample of the first few "chapters" of course material (including videos, reference learning scaffolding, and online exercises) can be found hosted at the Flat World website via: http://bit.ly/SwiftCourse1-2