Teaching the Fundamentals of Computer Programming: A Flipped Classroom Approach

Deena Engel, Clinical Professor of Computer Science, NYU
Craig Kapp, Clinical Associate Professor of Computer Science, NYU

Abstract

In this article, we report on our experience in developing a flipped classroom version of our Introduction to Programming (CSCI-UA.2) course, currently taught in Python. We address our motivation for the project; planning and preparation; the techniques and resources we used for creating the course content; our experience in the classroom; and our results. We share our experience with the larger goal of contributing towards ongoing research on best practices in using the flipped classroom approach in Computer Science undergraduate education and we seek to provide a model for developing this approach in a cost-effective way.

Introduction

Those of us who teach Computer Science are fortunate in that for us, developing technology-enhanced education tools and materials pertains to both the medium of delivery and to our subject matter. We are therefore in the unusual position as educators to use our subject field to inform both our course materials and our technology-based designs for education.

Teaching Computer Science programming courses using the flipped classroom model (also sometimes referred to as the “inverted classroom”) has seen a significant increase since 2000[1]. A great deal of research has followed in assessing the results for different student groups and for students and instructors over time. The benefits that a flipped classroom can bring include[2]:

- increases in learning performance
- positive attitudes

[1] Giannakos, M. and John Krogstie, 2014; Horton, Diane et al, 2014; Rutherfoord, Rebecca and James K. Rutherfoord, 2013; Heines, Jesse (Moderator) et al, 2015; Herala, Antti et al, 2015; Campbell, Jennifer et al, 2014; Maher, Mary Lou et al, 2015.
[2] Giannakos, M. and John Krogstie, 2014

- increased engagement
- more discussions (qualitatively measured)
- support for cooperative learning
- better learning habits

There are many aspects of a flipped classroom model which we will explore here including the nature and content of the course materials designed for self-study; the role of peer instruction in our model; the use of video and online exercises; the role of self-administered quizzes; and supporting the students' comfort levels in the classroom.

One group of authors states:

> *A number of suggestions for further research have emerged from reviewing prior and ongoing work on flipping the classroom. One recommendation for future researchers is to clearly describe the flipped classroom approach by providing detailed information for the materials used, as well as the pedagogical strategies, especially in subjects like IT/CS where technology sometimes has both the role of the content and the medium. This will allow us to identify which aspects, technologies, and concepts of the flipped classroom work better than others and to form best practices, providing a springboard for other scholars.*[3]

It is our hope that this article will address and support the need for a clearly described pedagogical and technological approach and contribute to the goal of designing and supporting best practices in Computer Science education.

Aims and Goals

We were motivated to develop a flipped classroom model for our Introduction to Programming course[4] in 2014 for a number of reasons: we wanted to use eLearning technologies to enhance and supplement our curriculum; to explore the "flipped classroom" model; to standardize course materials across many sections; to strengthen our student contact hours; to be able to better support larger class sizes; to provide additional support to our many instructors; and to do all of this with increased success for our students[5].

Standard course format

The course format in our department consists of two lectures per week of 1.25 hours each. Peer-tutoring is provided for introductory courses at scheduled hours in a public location on

[3] Giannakos, Michail N. and John Krogstie, 2014
[4] CSCI-UA.2, currently taught in Python; Syllabus Fall 2016: http://cs.nyu.edu/courses/fall16/CSCI-UA.0002-002/common_syllabus/
[5] Wilson, Brenda C. and Sharon Shrock, 2001

campus (typically in a computer lab). The course requires two midterm exams of one hour each held during class time; and a final exam of 1 hour 50 minutes held during the university exam week along with weekly or bi-weekly programming exercises. All sections of each introductory course require the same textbook; however, each instructor writes his or her own programming assignments, exams, class notes and class programming exercises. This results in duplication of effort on the part of the instructors, thus decreasing available time to meet with students.

Course format for our flipped format for Introduction to Programming

After turning to research results on the pedagogy and development of the flipped classroom in undergraduate Computer Science education[6], we developed an approach that supplements our traditional approach with in-class PI (peer instruction); the use of videos for class preparation; the "workshop days"; and insuring that course resources are fairly available to all students.

Peer instruction has been studied and shown to be effective in introductory Computer Science coursework[7]. One author describes PI as follows:

> *PI supports a student-centered learning environment by replacing some of lecture time's traditional "sage on the stage" activity with "guide on the side" student-focused activity.*

The use of videos for students' self-study and to supplement readings has been studied and showed to be effective[8], especially when it is paired with auto-graded quizzes that give immediate feedback for a small percentage of the final grade (5% in our case). The quizzes act both as an incentive and they help a student to assess his or her understanding of the material *before* the first class period on the topic[9].

In our flipped model, the students study from an online module[10], which contains videos, online programming exercises, and text each week before class. These course materials are hosted on our department web server and are now freely available under a GNU License[11]. Students take a quiz of ten questions (which are randomly selected from a relevant pool of questions)

---

[6] Horton, Diane et al, 2014; Giannakos, Michail N. and John Krogstie, 2014; Campbell, Jennifer et al 2014; Largent, David L. 2013; Lockwood, Kate and Rachel Esselstein, 2013; Maher, Mary Lou et al, 2015.

[7] Horton, Diane et al, 2014; Porter, Leo et al, 2013; Skirpan, Michael and Tom Yeh, 2015.

[8] Giannakos, Michail N. and John Krogstie, 2014; Horton, Diane and Michelle Craig, 2015; Campbell, Jennifer et al 2014; Largent, David L., 2013; Lockwood, Kate and Rachel Esselstein, 2013; Maher, Mary Lou et al, 2015.

[9] Horton, Diane and Jennifer Campbell, 2014; Herala, Antti et al, 2015; Lacher, Lisa L. and Mark C. Lewis, 2015; Campbell, Jennifer et al 2014; Maher, Mary Lou et al, 2015.

[10] Our course materials have been released under the GNU license and can be found at http://cs.nyu.edu/elearning/CSCI_UA_0002/

[11] http://cs.nyu.edu/elearning/CSCI_UA_0002/copyright.php

before the class meets; these quizzes are auto-graded with immediate feedback in our university's LMS. The first of the two weekly classes is spent in discussion and programming exercises with the instructor and the whole class. The second of the two weekly meetings is our "Workshop Day" format when the students work alone or in pairs, as they prefer, on a series of programming assignments while the instructor and more advanced undergraduate Computer Science students are available to assist, to role model, and to discuss the students' work. (A schedule from the fall 2016 semester is posted here[12].)

In addition, we set up an online repository of course materials for all of the instructors to draw from; this portion is further discussed in Section 4.2.

Preparation

Integrating a new teaching approach into an undergraduate department with many programs and courses requires planning and time. We first proposed this plan to our Chair in the fall, 2013 semester with a goal of starting development in the summer of 2014. We typically teach 8-12 sections of this course each fall and spring semester and at least two sections in the summers. We used the first summer in our timeline for course development; both semesters in the first academic year to run pilot sections; and both semesters in the second academic year for a full roll-out.

Semester and Year	**Development Phase**
Spring, 2014	Budget confirmation and planning materials in place
Summer 2014	Develop course materials (which were designed by the three pilot instructors)
	First Academic Year Begins
Fall 2014	Teach three pilot sections using a "flipped classroom" approach out of a total of eleven sections.
January, 2015	Modify the course materials based on feedback from the fall semester. These changes were typically small changes in clarifying specific aspects of the text and reflecting specific suggestions from students from the feedback that we received on each module.
Spring, 2015	Teach three pilot sections using a "flipped classroom" approach out of a total of nine sections.
Summer, 2015	Modify the course materials: These modifications included updates from student feedback along with closed captioning (for students with

[12] http://cs.nyu.edu/~kapp/courses/cs0002fall2016/schedule-tr.php

	disabilities as well as for students working in noisy environments such as during a long commute) and other changes. Prepare the Instructor Resource website.
	Second Academic Year Begins
Fall, 2015	Full implementation: all twelve sections of the course used the "flipped classroom" approach.
Spring, 2016	Our full implementation continued and all nine sections of the course used the "flipped classroom" approach. In this semester, two sections had over 100 students each as we piloted the "flipped classroom" approach in the large classroom environment.

Building the Course Materials

Curriculum

In order to ensure that all of our instructors' priorities were represented we conducted informal conversations with the group to select the "core" topics we all felt would be required so that students would be successful in any of the next level courses. The ten topics we decided to include are:

1. Getting Started with Python, Variables, Statements, Simple Input & Output
2. Data Types, Operators & Math Expressions, Formatting & Debugging
3. Boolean Logic, Conditional Statements & Using External Modules
4. Condition Controlled Loops
5. Count Controlled Loops & Nested Loops
6. Writing & Using Functions
7. Strings, String Manipulation, Sequences & Slicing
8. Sequence Structures & Lists
9. Exceptions & File Input/Output
10. Dictionaries

Technology

In terms of content delivery, our overarching goal was to construct a web-based environment that would be accessible on a wide variety of devices, including smartphones, tablets, Chromebooks as well as traditional PCs and Macintosh computers. We also wanted to ensure that our content would be fully accessible to students with disabilities.

Our department is fortunate to have an incredibly supportive IT systems group, and we were provided with hosting space on a secure, high-performance web server. We had no specific

requirements in terms of back-end software: our site was built using standard open-source tools that are available on most web servers (Apache and PHP).

We then turned our attention to the web interface; the videos; the interactive programming portions of the website materials; and the quizzes and assessment. Each of these is described in detail below.

Web interface

We based the design of all of our web-based materials on the popular "Twitter Bootstrap"[13]open-source front-end framework. In addition to giving us a "professional" looking site, Twitter Bootstrap also gives us the ability to ensure that all pages would be rendered in a "responsive" manner – i.e. content would resize automatically based upon the available screen size of a client's machine. In addition, we have found that web pages written using this framework are easily read by popular "screen reading" software packages that are used by students with visual disabilities.

Next, we turned our attention to the materials we wanted to deliver to our students and decided that our modules would include a mixture of the following types of content:

- Descriptive text and code samples
- Short videos to illustrate key concepts
- Interactive programming exercise
- Summary quizzes

Descriptive text and code samples were fairly straightforward and could easily be written using standard HTML / CSS code. Before doing this we created a "style guide" that would be used to create a consistent "look and feel" throughout our modules (i.e. green boxes for "Sample Programs", red boxes for "Programming Challenges", etc.). Twitter Bootstrap simplified this approach as we used its impressive array of CSS base-classes as the basis for our design.

Videos

Video content is embedded in each module, and each module contains multiple videos. We decided to use a YouTube account (provided to us through our university's Google Apps for Education[14] program) for a variety of reasons: YouTube is incredibly easy to use; it has no file size quotas; and it is backed by Google. YouTube has an excellent track record in terms of

[13] http://getbootstrap.com/2.3.2/
[14] https://www.google.com/edu/

uptime and availability and it offers the ability to easily incorporate closed captions into video content.

In terms of recording videos we focused on developing a series of short segments that each focused on a single concept. We decided that this content would mainly consist of screen recordings and audio tracks, rather than a visual depiction of the instructor. Our goal here was to record quick, succinct tutorials that would cover specific "mechanical" topics that did not necessarily need to be covered in an in-class lecture session.

To do this we used the "Camtasia Studio[15]" software suite by TechSmith, which is an easy-to-use commercial solution for screen recording. Using a standard laptop computer and USB microphone / headset, we were able to record, edit and produce all of our video content without the assistance of a professional recording studio or video editing team. The tutorial content that comes shipped with Camtasia provided us with enough training to get started with the product, and in the end we were quite pleased with the overall video production workflow.
It's worthwhile to note that there are free / open-source alternatives to Camtasia, including the Open Broadcaster Project[16], Apple Quicktime[17] and ShareX[18]. We elected to use Camtasia mainly because it offers an integrated recording and editing interface which allowed us to rapidly produce videos without having to switch between multiple software packages, but our videos could have easily been produced using a combination of free / open-source alternatives.

By the time we had finished developing our modules we had recorded and edited over 70 video units. The recording process we used included the following steps:

1. Write a script: We found that videos written using scripts tended to be of much higher quality than those that were recorded "off the cuff".
2. Set up your environment: All of our videos consist of recordings of a computer screen along with the voice of an instructor. In order to produce this kind of video we needed a quiet place to work and to set up our computers so that only our programming interface was visible.
3. Record: Using Camtasia Studio we would initiate a recording session and walk through our script at a measured pace. If we encountered a small issue (i.e. stumbling over some words, etc.) we could easily pause and know that those issues would be removed in post-production. If the issue was large enough we would stop the recording completely and re-start from the beginning. We made sure to include a variety of both

[15] https://www.techsmith.com/camtasia.html
[16] https://obsproject.com/
[17] https://support.apple.com/en-au/HT201066
[18] https://getsharex.com/

male and female voices in our videos to ensure that students were not always hearing the same instructor in each module.

4. Edit: We used Camtasia studio to trim out any “ums” and “ahs” in our audio track. We added in any special effects at this time, including overlays and camera zooms.
5. Produce: The video was produced and saved as a valid video file (usually an MP4 video).
6. Upload: The video was transferred to our university YouTube channel. Once this was complete we provided metadata on the video, including which module it was associated with.
7. Caption: Finally, we used YouTube’s built-in closed captioning system[19] to caption the video in English to ensure accessibility of our materials. This system works fairly well and captions approximately 80% of the video correctly; corrections were done manually.

Interactive programming portions of the website materials

We also wanted to incorporate interactive programming challenges directly into our modules. Our goal was to give students the ability to quickly test their newly acquired programming skills right away without having to leave their web browser. To do this we used the open-source Skuplt[20] framework. This tool allows users to write standard Python code in their web browser using an interface written in HTML. Because Python does not run natively in a browser, Skulpt converts Python code into JavaScript which is then executed by the browser. This gives students the ability to experiment with programming concepts directly and not have to worry about launching an external IDE.

Quizzes and assessment on the modules

When students finish with a web-based module we need a way to easily assess their learning and store their results. We also wanted to build in a system that would allow students to test their knowledge and identify concepts that may have been difficult for them. In order to do this, students are asked to take a short quiz to demonstrate their understanding of the topics upon completion of every module. We deliver these quizzes to students using our university LMS[21] as this is the only system available to us that lets us safely associate student grades with their university ID in a FERPA compliant manner. Each instructor decides how many times the students may be permitted to take the quiz on a given module and whether to store the highest grade or the most recent grade. This process is centrally managed and does not need to be completed by every instructor.

---

[19] https://support.google.com/youtube/answer/2734796?hl=en
[20] http://www.skulpt.org/
[21] Our university LMS is based on Sakai: https://sakaiproject.org/

IMPLEMENTATION

Classroom support

Classroom support has proved to be an integral part of the work we are doing with this course. With only 2.5 hours of student contact per week, we need to provide students with as much guidance as possible during the in-class hours that we have. To this end, we found that a ratio of approximately 25:1 (twenty-five students per teaching assistant) works best on the in-class programming days.

Our teaching assistants are typically undergraduate Computer Science majors; Computer Science minors; or students in our Web Development minor[22] programs. We observed that our student tutors typically gain a great deal of confidence from tutoring under our guidance. We also hire a graduate student to mentor and assist us with managing the many undergraduate teaching assistants. We have found that our tutoring mentor develops a good working relationship with our teaching assistants and is able to help them coordinate schedule changes when needed, answer questions on the course content and meet when concerns arise. This additional level of support is cost-effective and extremely helpful given the large class sizes.

On in-class programming days, we often start the class with a brief, 10-15 minute, review of the concepts and an introduction to the programming problem set in 10-15 minutes. The students work on their projects for just under an hour and we still have 5-10 minutes at the end of the class session to wrap up and discuss questions that were common to many students or make announcements relevant to the materials or the course.

Instructor support

Many of the instructors who teach “Introduction to Computer Programming” are part-time adjunct faculty members, and we feel that it is our duty to provide them with as much support as possible so that they do not feel “burnt out” by the course development process.

In addition, we seek to standardize this course so that we can be sure that all students who successfully complete this course will come out with a known skill set and will be well-prepared for the next course(s) in our sequence of study.

One resource that we have produced in order to help support our faculty members is a comprehensive “instructor-only” website that contains the following materials:

[22] http://cs.nyu.edu/home/undergrad/minor_programs.html

- A common course syllabus complete with departmental policies, required topics of study, grading criteria, etc.
- Sample schedule of course meeting dates, including suggestions for when to schedule “lab days”
- Links to all self-paced learning modules
- Sample homework assignments
- Exam resources, including sample exams for in-class practice
- A fully customizable website that adjuncts can download and use “out of the box” in lieu of developing a course website from scratch
- Directions on how to use the university LMS for grading, assignment collection and setting up weekly quizzes

We also provide our faculty members with a variety of ways in which they could seek out support in person. During our pilot year we blocked off time for weekly “drop-in” meetings with instructors who had questions about the flipped classroom project; we found that instructors appreciated having the ability to stop by and informally chat with their colleagues about how the course was going. This year we have shifted our instructor support mechanism to e-mail and Skype as well as face-to-face meetings that are scheduled based on the instructor’s availability.

Integration with the LMS

The university LMS serves as the backbone for most day-to-day administrative tasks associated with this course. Due to the fact the LMS operates as a secure, FERPA compliant space, all student-identifiable material is housed within this system. With that said, schools without an LMS could replicate the quizzing functionality of our model using another product (Google Forms, Survey Monkey, etc.) or even through paper-based quizzes.

Currently the university LMS provides us with the following capabilities:

- Secure collection of student homework, including the enforcement of any “late submission” policies put in place by an instructor
- A dynamic grade book that can be accessed by students, instructors and teaching assistants
- A quizzing system that administers and scores weekly quizzes and stores the results in the course grade book
- A messaging system that lets an instructor easily e-mail his or her class to communicate due dates, schedule changes, etc.

We have found that the university LMS has served as a crucial piece of technology that effectively provides centralized space for student recordkeeping.

Challenges

While the overall implementation of our “flipped classroom” went relatively smoothly, there were some challenges we had to address along the way. First and foremost, we feel it is important to not underestimate the amount of time that was put into the development of the course materials which were authored by two full-time faculty members and a full-time student worker over the course of the summer session (3 months). Anyone attempting to replicate this model should be prepared to devote a significant amount of time to this process, especially at the beginning.

Interviewing, hiring and scheduling TAs has been challenging as well since our student population is always in flux. Given the size of our classes we routinely need to hire at least 15 TAs that have varied schedules in order to provide coverage for all course sections. Dealing with this process takes up a significant amount of time at the beginning of each semester.

We have also experienced some challenges as we have tried to adapt our workflow to that of the LMS. One of the biggest issues we have experienced has been interoperability with our weekly quizzing system on the university LMS. During the course development phase, we designed hundreds of questions that could be used in a series of “question pools” that would be used to dynamically construct unique quizzes for each student. However, we quickly found that while this was technically possible, the system was set up in such a way as to remove all HTML formatting from our quizzes (which is especially disruptive to Python code in which the code layout is meaningful). The result is that we have used a variety of workarounds, including hiring undergraduate students to input and correct the quiz questions by hand.

Working with large class sizes

In general, sections of “Introduction to Computer Programming” are limited to a maximum enrollment of 60 students. However, in the spring, 2016 semester, we had an opportunity to test our newly developed “flipped classroom” model on two large sections taught by the authors of this article. Each of these two sections eventually saw enrollments in excess of 115 students.

In order to maintain a high level of peer interaction in these two “jumbo” sections we made a concerted effort to hire additional peer tutors so that we could continue to support our desired 25:1 tutor-to-student ratio. We asked these tutors to circulate around the room during “workshop” days, but after a few weeks we found that each tutor ended up gravitating toward specific clusters of students. This worked out very well for all parties involved: students benefited by having dedicated support from tutors with whom they had been building

relationships with over the course of the semester, and tutors were able to tailor their support to the individual needs of their students. In addition, we found that many of these students would seek out their particular tutors during that tutor's "drop-in" office hours to ask follow up questions and get general support in the class.

The instructors of the course also offered additional student office hours throughout the week, including office hours held in a large conference room. This helped to serve as a group meeting space where the instructor would walk around the room and assist students with individual issues. Students who did not have questions were welcome to come and work in the community space as well, and we found that many times students ended up helping each other while the instructor was working with another student.

During the "lecture" portion of class we found that keeping track of how well students were grasping key concepts became quite challenging – it's incredibly difficult to have a meaningful discussion with 120 students at the same time. To assist with this one of the authors developed a live quizzing tool called the "Interactive Classroom." Using this tool, we were able to poll students using a variety of question types and aggregate their responses in real-time. Once the responses were collected they would be anonymized and projected to class, allowing the instructor to focus in on specific types of issues based on the individual needs of the students. The system also served as a way to take attendance in class, as all students were required to log into their university account prior to participating in one of these quizzes.

One of the biggest challenges in managing such a large number of students was the grading of our paper-based exams. We were incredibly worried about how this could be done at such a scale, but eventually we found a free online service called "Gradescope[23]" that solved the issue for us.

Gradescope is an online tool that is designed to make it easy for instructors to grade paper-based exams using a simple web-based interface with scanned PDFs of the students' exams. We have found that Gradescope provides a level of consistency in grading that is very difficult to achieve using paper-based exams. We have also found that cheating by "erasure" has been eliminated using this process - since the exams are scanned and returned to students digitally it is impossible for them to use a pen or pencil to change a particular answer after the exam has been returned and claim that it was incorrectly graded.

[23] https://gradescope.com

Department support

We have received a great deal of support from our department for a number of reasons. First, we have been able to develop these materials at a nominal cost. Since our videos consist of voice-over and screen capture, we did not use a studio or expensive environment to produce them but rather worked at home or at our desks. Providing web delivery of the course content, along with our internal web-based repository for the instructors allows for shared materials and saves a great deal of the instructors' time. The teaching community benefits as we have made the open source materials available so that others might benefit as developing the materials is a one-time cost which could discourage other departments[24]. In addition, we have succeeded in increasing our class size with little incremental cost and less need for further classrooms by changing how we use the time spent in our classrooms. We have been able to increase the volume of homework three-fold or more over our traditional course model so that students gain much more practice in programming and are exposed to many more types of programming problems in their first semester. We have raised the level of difficulty of this course significantly and as a result, we have recently had to re-write our placement exams to reflect the more challenging material that we now handle. Finally, we have found benefits for the instructors: the "flipped classroom" model allows us to integrate new instructors more easily and we all benefit from the additional student contact and the satisfaction that comes from working closely with students and watching them grow and learn.

We have received positive support from the college as we have been invited to speak at several panels on technology and education and the authors jointly received a 2016 "Teach/Tech" award from the College of Arts and Science[25] at NYU.

Feedback and results

Anecdotal evidence from our pilot study suggested that students enjoy the "flipped" format of the course and appreciated the interactive materials that have been developed. Some sample comments gathered from course evaluations during these pilot sections included the following:

- "The online material is extremely helpful."
- "The modules were super helpful and I really appreciated being able to use them to learn material. They were also really interactive (challenges and examples) and helped me apply the concepts I had just learned. I had a lot of fun and learned a lot from this course! "

[24] Baldwin, Douglas 2015; Campbell, Jennifer et al, 2014;
[25] http://cas.nyu.edu/page/teaching.awards

- “The structure of the class forces me to learn because we do programs during class rather than only for HW. This is my first compsci class and I am graduating in May but I'm extremely happy I decided to take this course during my last semester at NYU.”
- “The online video tutorials and quizzes are extremely helpful to the course and is what makes Kapp stand out as a professor.”
- “Please keep the video and quiz modules. Very helpful and straightforward. Very useful for review and for preparation for that day's lecture.”
- “Modules and lectures are very helpful.”
- “Useful to program in class. Good lectures and set of notes online.”
- “I have nothing but positive reviews for the class, the tutor, and the structure of his teaching. Thank you so much!”

In addition to end of semester evaluations, we also ask students to anonymously rate each of self-paced learning module on an ongoing basis. Students rate the modules on a scale of 1-5 and also have the ability to provide free-form feedback on the quality of the materials presented. For the Spring 2016 semester the average rating for our modules was 4.28 out of 5.00. This information is actively used at the end of each semester to update and refine our materials.

The authors of this paper are also actively working on developing a formal system to assess the efficacy of the program by looking at how students in our pilot sections perform in the next level course (CS101) as compared with their peers who were taught in a “traditional” section. We found that students who participated in one of our “flipped” pilot sections performed about half a letter grade better than their peers who participated in a “traditional” section of the course. With that said, we feel that our sample size was too small to come to any concrete conclusions, but we do believe that the initial results are promising and we look forward to continuing this analysis at the end of the next academic year.

Future Plans

Future plans include ongoing development of the Interactive Classroom software to complement the “lecture days”; to continue to develop original and challenging course materials, especially new assignments and problem sets; and to work with other faculty to develop “flipped classroom” materials for additional Computer Science courses in our institute.

With respect to research, we plan to continue to gather data from our current courses and beyond in order to follow students as they progress through the Computer Science major and minor programs. We plan to collect and tabulate both qualitative and quantitative feedback and data from our students in order to determine whether there is a long-term impact of their flipped classroom experience and how such an impact might be described and measured.

We wish to thank Professor Joanna Klukowska for her helpful feedback as we developed this article.

Bibliography

Douglas Baldwin. 2015. Can We "Flip" Non-Major Programming Courses Yet?. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 563-568. doi:http://dx.doi.org/10.1145/2676723.2677271

Jennifer Campbell, Diane Horton, Michelle Craig, and Paul Gries. 2014. Evaluating an inverted CS1. In *Proceedings of the 45th ACM technical symposium on Computer science education*(SIGCSE '14). ACM, New York, NY, USA, 307-312. doi:http://dx.doi.org/10.1145/2538862.2538943

Michail N. Giannakos, John Krogstie, and Nikos Chrisochoides. 2014. Reviewing the flipped classroom research: reflections for computer science education. In *Proceedings of the Computer Science Education Research Conference* (CSERC '14), Erik Barendsen and Valentina Dagiené (Eds.). ACM, New York, NY, USA, 23-29. doi:http://dx.doi.org/10.1145/2691352.2691354

Jesse M. Heines, Jeff L. Popyack, Briana Morrison, Kate Lockwood, and Doug Baldwin. 2015. Panel on Flipped Classrooms. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 174-175. doi:http://dx.doi.org/10.1145/2676723.2677328

Antti Herala, Erno Vanhala, Antti Knutas, and Jouni Ikonen. 2015. Teaching programming with flipped classroom method: a study from two programming courses. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli Calling '15). ACM, New York, NY, USA, 165-166. doi:http://dx.doi.org/10.1145/2828959.2828983

Diane Horton, Michelle Craig, Jennifer Campbell, Paul Gries, and Daniel Zingaro. 2014. Comparing outcomes in inverted and traditional CS1. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (ITiCSE '14). ACM, New York, NY, USA, 261-266. doi:http://dx.doi.org/10.1145/2591708.2591752

Diane Horton and Michelle Craig. 2015. Drop, Fail, Pass, Continue: Persistence in CS1 and Beyond in Traditional and Inverted Delivery. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 235-240. doi:http://dx.doi.org/10.1145/2676723.2677273

Diane Horton and Jennifer Campbell. 2014. Impact of reward structures in an inverted course. In*Proceedings of the 2014 conference on Innovation & technology in computer science education*(ITiCSE '14). ACM, New York, NY, USA, 341-341. doi:http://dx.doi.org/10.1145/2591708.2602671

Lisa L. Lacher and Mark C. Lewis. 2015. The Effectiveness of Video Quizzes in a Flipped Class. In*Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 224-228. doi:http://dx.doi.org/10.1145/2676723.2677302

David L. Largent. 2013. Flipping a large CS0 course: an experience report about exploring the use of video, clickers and active learning. *J. Comput. Sci. Coll.* 29, 1 (October 2013), 84-91. Accessed October 5, 20116.

Kate Lockwood and Rachel Esselstein. 2013. The inverted classroom and the CS curriculum. In*Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 113-118. doi:http://ezproxy.library.nyu.edu:2079/10.1145/2445196.2445236

Mary Lou Maher, Celine Latulipe, Heather Lipford, and Audrey Rorrer. 2015. Flipped Classroom Strategies for CS Education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 218-223. doi:http://dx.doi.org/10.1145/2676723.2677252

Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 177-182. doi:http://dx.doi.org/10.1145/2445196.2445250

Rebecca H. Rutherfoord and James K. Rutherfoord. 2013. Flipping the classroom: is it for you?. In*Proceedings of the 14th annual ACM SIGITE conference on Information technology education* (SIGITE '13). ACM, New York, NY, USA, 19-22. doi:http://dx.doi.org/10.1145/2512276.2512299

Michael Skirpan and Tom Yeh. 2015. Beyond the Flipped Classroom: Learning by Doing Through Challenges and Hack-a-thons. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15). ACM, New York, NY, USA, 212-217. doi:http://dx.doi.org/10.1145/2676723.2677224

Brenda Cantwell Wilson and Sharon Shrock. 2001. Contributing to success in an introductory computer science course: a study of twelve factors. In Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (SIGCSE '01). ACM, New York, NY, USA, 184-188. doi:http://ezproxy.library.nyu.edu:2079/10.1145/364447.364581