CS Education Research Summit – White Paper

Flipped, Crowd-Sourced, Gamified, Socialized and Funkified: Throwing Everything But the Kitchen Sink at Introductory Programming Classes

Celine Latulipe, Mary Lou Maher, Bruce Long, Audrey Rorrer
Department of Software and Information Systems
College of Computing and Informatics
UNC Charlotte
clatulip@uncc.edu

The number of alternative teaching methods being explored in Computer Science (CS) education is exploding. Active learning, online learning, hybrid learning, social learning are just a few of the popular methods being discussed, tried, and adapted for teaching CS. The motivation for trying these approaches is often to respond to new technology and/or research on learning. Yet, we are starting to see that research in CS education has its own characteristics and associated challenges. In this white paper, we focus on teaching CS subjects that include the development of programming skills. We highlight the difficulties in traditional approaches to teaching programming courses and the challenges in introducing new teaching methods and understanding the effect of interventions.

Active learning is unarguably more effective than passive learning. For example, the finding that retrieving from memory improves long term retention [12] is an indication that students are engaged in the most effective learning when they are actively retrieving from memory, not when they are listening to a lecture. We are seeing that new approaches to teaching programming, such as the flipped classroom method, students spend time in the classroom working on activities, such as solving problems, writing code, or discussing concepts with their peers. This creates a learning environment where students learn by interacting and collaborating with their peers. Many studies have shown that working with peers in classroom activities such as programming exercises improves overall learning, increases confidence in students and makes coding fun [1-7]. Classroom quizzes have also been used as an effective mode of instructor-student interaction and many new technologies including 'Clickers' have successfully been used to foster learning by questioning in classrooms [8]. Since the earliest attempt to use a flipped classroom method to teach an economics course [6], educators have used this teaching method in many courses including Mathematics [7], Biology [9], Business Management [11], Industrial Engineering [13] and Computer Science [5]. The flipped classroom teaching method has been used in both small courses of around 40 students as well as in large courses with hundreds of students. In large classrooms however, the students were divided into smaller groups when they were working on class activities [11]. Flipped courses require students to come to the class after viewing the required materials so that they can participate in the class activities. These learning materials, such as lectures and demonstrations, are delivered to students through an alternate medium such as podcasts or video-on-demand services [11, 13]. Students can view these materials on their own time and at their own pace prior to coming to the class where they then engage in various structured class activities.

This shift in teaching CS to more active learning, particularly for courses that include programming, has the potential to address many of the problems that the traditional approach to education has when learning how to program. While many instructors are trying various approaches to active learning in CS, very few are doing research in CS education. Often, the interest in adopting and adapting new teaching methods in CS is driven by novel technology and a passion for teaching, not with a goal of contributing to research in education. A shift from teaching to research about education is a big one, and has its own challenges. This white paper introduces three research questions that focus on challenges for CS research in teaching introductory programming, based on our experiences in trying new teaching methods in a Programming I course and the Web Applications Programming course.

1. Why do so many people fail or perform poorly in intro programming?
2. What interventions can we try to improve student performance or help students learn?

3. How do we evaluate if the intervention(s) worked?

R1: Why do so many people fail or perform poorly in intro programming?
The average drop/fail/withdraw rate for the traditional Programming I course has been 39% and for the Web Applications Programming course has been 10% over the past few years at UNCC. This rate is fairly typical for large urban universities like UNCC (Drexel reported rates decreasing from 49 to 38% as part of a Pew Project course redesign [12]). For students who do complete the course and continue on in the CS or IT majors, we hear complaints from faculty (especially those who teach courses such as data structures and mobile or server application development) that many students are not proficient enough at programming, and do not seem to be self-sufficient learners when it comes to picking up new languages or programming environments. Based on our experience with the students at UNC Charlotte, we believe that the traditional approach to teaching programming does not work for many students who are new to programming for the following reasons:

- Traditional programming problems based on calculating or accounting problems are boring.
- Presenting programming concepts via lecture is ineffective, because it is out of context.
- Lectures are isolating and don't allow enough social engagement around the material.
- Girls paired up to program with boys don't learn because the boys tend to take over.
- Students don't get enough fun, short programming exercises with immediate feedback.
- Tests involving writing code on paper are unrealistic and don't evaluate learning.

However, these observations are based entirely on our experience, our interaction with students that do not do well, and some conjecture. We act on these observations and conjectures by introducing new ways of teaching, new content for programming exercises, and explore ways to make programming fun. While this may be effective, it is not research. This research question highlights that we know there are reasons why traditional introductory programming courses do not serve many of the students well, but we do not have a strong methodology for identifying and acting on these reasons.

R2: What interventions can we try to improve student performance or help students learn?
There are so many educational innovations in the literature and promoted online that it can be hard for an educator to decide which approaches to adopt when the traditional methods for teaching programming don't work. At UNC Charlotte, the Department of Software and Information Systems has been on a mission to revolutionize our teaching. We have a goal of flipping all of our classes eventually, beginning with introductory programming and HCI classes. However, we are not just flipping the class, we really are attempting to incorporate every promising teaching methodology that we think can complement the flipped class structure, and we are trying a few tricks of our own. Our objective is simply this: increase student engagement in order to increase student performance.

The following is a list of interventions that we have deployed in our Programming I and/or Web Applications Programming courses:

- **Flipped classroom**: Before coming to programming labs, students watch videos (crowd-sourced by other students) that teach core programming concepts, and they do the assigned reading and practice programming. They take a quiz, either before class or at the beginning of class, to ensure they have done the prep work. They are also encouraged to critique the videos and suggest other videos that cover the same material, encouraging them to take ownership over the sources of knowledge and recognize that knowledge is all around them.
- **Pair Programming**: Students engage in pair programming in the lab (in CSI girls are only paired with girls), and use an AutoGradr system that gives them immediate feedback about whether their coding has been successful. Lab partners are assigned, but change every two weeks, giving students enough time to get to know one another, but not so much time that a poorly-performing partner adversely affects a students' grade.
- **Media Computation**: Students learn programming through manipulation of multi-media elements, which are highly engaging and visual in nature. Students are encouraged to use their own images and create outputs that are personally meaningful.

- **Question Tokens**: Students are given special 3D printed tokens when they arrive at the lab, and must give up their token to ask an instructor a question. This prompts them to work hard to find the answer on their own, ask each other and look things up, promoting self-sufficient learners and problem solvers. They work hard in order to not give up their tokens.
- **Peer Instruction Workshops**: Instead of lectures, we hold peer instruction workshops that consist of challenging quizzes where students work together in teams to figure out the answers to the quiz questions. This is active learning, and students are often asked to 'find someone who has a different answer than you and convince them you are right'. We often ask similar questions two or three times in a row, so students have a chance to try out what they have just learned and cement the knowledge.
- **Lightweight Teams**: Students are divided into teams of 5 at the beginning of the semester and sit with these teams during the peer instruction workshops each week, in assigned seating. They work together on quizzes and on paper problem solving activities. Teams compete to earn the most badges.
- **Badges**: Individuals earn badges for various achievements, such as completing Lab bonus questions, adding concepts to the course glossary, sharing class notes, etc. A team's badge count is simply the sum of badges earned by all team members. Badges don't affect student grades, but the teams with the most badges at the end of the semester win prizes.
- **Tests of Understanding**: We have 5 lower-stakes tests, rather than one or two high-stakes exams, and these tests involve (multiple-choice, scan-tron) questions that are designed to evaluate student's understanding of code and coding concepts. We don't believe that asking students to write code on paper is a realistic way to test whether a student has learned computer programming.

We are reacting to our observations by trying various interventions, which we recognize does not constitute "research" in CS education. As a research question we are interested in how we can correlate a specific intervention to address a specific challenge or learning outcome. This research question asks how can we identify and associate interventions for specific learning needs.

R3: How do we evaluate if the intervention(s) worked?

This is possibly the most difficult question for CS education research. A robust research approach to evaluating interventions is to introduce them one at a time, and to have a control group. The interventions should be varying a single variable and we should minimize confounding variables. We should be collecting all data from the experiments that may be relevant to the effect of the intervention.

Unfortunately, control groups may be inappropriate in an educational setting, where it would be unfair not to offer pedagogical interventions that are likely to enhance student engagement and learning effectiveness to everyone., Waiting for scientific evidence to support particular interventions would be detrimental to our student body. So, if we continue to do as we have done with our classes, and incorporate a variety of seemingly complementary pedagogical tactics, how do we evaluate their success? Our approach has been to gather detailed feedback from the students throughout the semester, use data analytics tools to scrutinize activity on the learning management system and to trust our own observations. The detailed feedback from the students is most useful in identifying how particular interventions are perceived as affecting engagement and learning. These are all 'within-semester' approaches to measurement. We also have the ability to examine the student grades and compare them to previous years, and watch the trends as we move forward with subsequent deployments. This long-term analysis is confounded by changing student demographics, so our future analyses will attempt to isolate these student demographic factors. The long-term data may be most useful in helping us understand whether the overall approach of combining flipped class structure with the media computation approach is effective. Finally, the most promising source of evaluation data will be in comparing the performance of students in this new version of the Programming I course to those in the Programming II course to determine if students from the flipped versions of Programming I perform differently than students matriculating in from traditional classrooms as they move through Programming II.

Feedback collected so far (48 voluntary responses from 91 students in our new CSI) has indicated that our interventions are working well. Our most surprising result is that the socialization elements have been very well received. This was unexpected because past experiences have often shown that students dislike group work and complain about group members. We characterize our socialization as light-weight, but long-term, and believe that this may be a key factor in increasing student engagement. Our assumption is that forcing small teams of students to sit together all semester gives students four new friends that they otherwise might have had trouble making in a large class. Because the teams are all packed close together, and sit in the same place each week, students also tend to make friends with members of nearby teams. The use of peer instruction quizzes, in which we often ask students to 'find someone who has a different answer than you and convince them that you are right', leads to teams debating other teams over particular aspects of the course content. The fact that the students earn badges together, work together on peer instruction quizzes and other low-stakes activities means that they are not concerned about a slacker team-mate strongly affecting their final grade. Our students overwhelmingly voiced appreciation of the team aspects of the class (4.3/5 average rating). Female students are also strongly in favor of same-gender lab work (4.0/5.0), though male students less so.

Our feedback indicates that the gamification aspects appeal particularly to the female students in the class, but are generally liked by both genders. This result also surprised us – we expected the gamification aspects to be more popular with male students. The response to the use of tokens to ask questions in lab is mixed – we think that though the tokens may be forcing the students to work harder to solve problems without help, this is very different for them. They may sense that it is good for them, but they may still not like it all that much. We feel that this intervention is one that my need fine-tuning in order to help students understand when the trade-off between keeping the token and asking a question should be tilted in favor of asking for help rather than losing an opportunity to understand something important.

We acknowledge that our current approach lacks the structure of a true experimental design, which limits the type of evidence we are able to present at this time. However, we believe that performing experimental research on CS education interventions prevents us from our responsibility as stewards in ensuring the success of all of our students. To conduct experimental education research would slow our progress in retaining qualified and interested students. There are viable alternatives to strict scientific inquiry that enable us to intervene immediately and to impact our students directly without creating disparate impact. We use mixed-methods quasi-experimental research, action research, and case studies, all of which provide data rich content to contribute to the ongoing conversation about CS pedagogical reform. This conversation asks for a better understanding of the methods for doing CS education research, the range of data that can be collected to evaluate interventions, and the acceptable qualitative and quantitative approaches to drawing generalizable conclusions from our intervention experiences.

References

[1] Gannod, G.C., Burge, J.E. and Helmick, M.T. 2008. Using the inverted classroom to teach software engineering. ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08 (2008), 777–786

[2] Gehringer, E.F. and Peddycord,III, B.W. 2013. The inverted-lecture model: a case study in computer architecture. *Proceeding of the 44th ACM technical symposium on Computer science education* (New York, NY, USA, 2013), 489–494.

[3] Guzdial, Mark. 2013. Exploring hypotheses about media computation. In Proceedings of the ninth annual international ACM conference on International computing education research (ICER '13). ACM, New York, NY, USA, 19-26.

[4] Hanks, B. 2006. Student attitudes toward pair programming. *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (New York, NY, USA, 2006), 113–117.

[5] Kaner, C. and Fiedler, R. 2005. Inside Out: A Computer Science Course Gets a Makeover. [2005]. www.kaner.com/pdfs/kanerfiedleraectprint.pdf

[6] Lage, M.J., Platt, G.J. and Treglia, M. 2000. Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment. The Journal of Economic Education. 31, 1 (Jan. 2000), 30–43.

[7] Lockwood, K. and Esselstein, R. 2013. The inverted classroom and the CS curriculum. Proceeding of the 44th ACM technical symposium on Computer science education (New York, NY, USA, 2013), 113–118.

[8] Mayer, R., Stull, A., Almeroth, K., Bimber, B., Chun, D., Bulger, M., Campbell, J., Knight, A. and Zhang, H. Using Technology-Based Methods to Foster Learning in Large Lecture Classes: Evidence for the Pedagogic Value of Clickers

[9] Moravec, M., Williams, A., Aguilar-Roca, N. and O'Dowd, D.K. 2010. Learn before lecture: A strategy that improves learning outcomes in a large introductory biology class. *CBE life sciences education*. 9, 4 (2010), 473–481.

[10] Roediger, H.L. and Karpicke, J.D. 2006. Test-enhanced learning: taking memory tests improves long-term retention. *Psychological science*. 17, 3 (Mar. 2006), 249–255.

[11] Schullery, N.M., Reck, R.F. and Schullery, S. 2011. Toward solving the high enrollment, low engagement dilemma: A case study in introductory business. *International Journal of Business, Humanities and Technology*. 1, 2 (2011), 1–9

[12] The Pew Learning and Technology Program Newsletter Vol. 4, No. 3 September 2002. http://www.thencat.org/PCR/PewNews/PLTP13.html

[13] Toto, R. and Nguyen, H. 2009. Flipping the work design in an industrial engineering course. *Proceedings of the 39th IEEE international conference on Frontiers in education conference* (Piscataway, NJ, USA, 2009), 1066–1069.