# Different Audiences but Similar Engagement Goals:

## In-Progress Work on Two Course Transformations

Donald Acton and Edwin M. Knorr
{acton,knorr}@cs.ubc.ca
Department of Computer Science
The University of British Columbia

## ABSTRACT

This paper reports our experience in transforming two undergraduate Computer Science courses at the *University of British Columbia* (UBC). In particular, we are applying an assortment of best practices from educational research known to increase student engagement. The two courses are being transformed in different ways because their learning goals and audiences vary greatly. For example, the courses use different programming languages; they differ with respect to the number, type and frequency of labs, tutorials, and class time; and one is an elective for computer scientists, whereas the other is a required course for non-specialists. Despite these differences, both courses have a greater active learning component compared to a "traditional" lecture, with one of them adopting a flipped classroom approach. To judge the success of these efforts, we conducted numerous surveys to determine changes in student attitudes and to identify what works—and what doesn't—for these courses.

## Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer Science Education, Curriculum.*

## General Terms

Classroom pedagogy, active learning.

## Keywords

Computer networking, C programming, data structures, teaching, learning, course transformation, student engagement, flipped classroom

## 1. INTRODUCTION AND BACKGROUND

Evidence from educational research on student learning in science strongly suggests that only a small fraction of the information presented in a traditional lecture is retained by a student at the end of the lecture [10]. (By "traditional", we mean an instructor: lectures for most of the class; writes notes on a blackboard, whiteboard, or PowerPoint slides; and has minimal student participation. Essentially the instructor is "a sage on a stage.") Based on the research for active learning [5][3] there has been a call to more actively *engage* students in the classroom. Unfortunately, research also shows that few science courses employ active learning. The challenge then becomes how to proceed.

In this paper we report on the approaches we used to incorporate active learning strategies in a major revision of one course (CPSC 317: computer networking), and in a new course (CPSC 259: data structures in C). These two courses can be viewed as "sandboxes" for developing implementation and active learning deployment strategies for our other courses. For example, because CPSC 259 is a fairly new course, and is a terminal computing course (i.e., it has no CPSC successor courses), we have more flexibility in the breadth and depth of course components from offering to offering. (e.g., responding dynamically to difficulties in the class and lab, and spending more time on important topics that students are struggling with). The rest of this paper is organized as follows:

- Sections 2 and 3 highlight the content of these courses. For each course we provide a basic course description, and the strategies and techniques being used to increase student engagement.
- In Section 4 we summarize our successes and failures, and provide guidance on what to watch for.
- In Section 5 we highlight how we are using our experiences from these courses to guide revisions in other courses.
- Finally in Section 6 we summarize our results and provide suggestions for future work.

## 2. CPSC 317: INTERNET COMPUTING

CPSC 317 is an elective course in computer networking open to anyone who has completed all five computer science courses comprising the first two years of our degree programs. The course explores the implementation details and issues associated with the design of the link, network, transport, and application layers of the TCP/IP protocol suite, while trying to provide insight into how the Internet works.

Variations of this course have been taught at our institution since the early 1990s. The original target audience was students interested in how computer networks worked—and were implemented. As a by-product, students learned the UNIX networking API and developed applications that used a computer network. That course was *not* specifically targeted at students whose primary interest was writing programs that *use* networks.

### 2.1 Course Content and Structure

Prior to the course's revision, it was a fairly standard course with two 80-minute lectures per week, a weekly one-hour tutorial session, 3-4 large programming assignments, and the requisite midterms and final exam.

Part of the motivation behind changing the course was the observation of a marked shift in the number and types of network-based applications and how these applications are developed. This shift has been spurred by increased data speeds and capacities in both the wired and wireless networking domains combined with the emergence of hardware like the iPhone, iPad and Android-based devices that thrive in these new networks. At the same time, cloud-based data sharing services like Dropbox, Wuala, YouTube, and Google Docs have been deployed. These services encourage

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*WCCCE '13*, May, 2013, North Vancouver, Canada.
Copyright © 2013 ACM XXX-1-4503-0098-8/10/05... $10.00

users to push their content to the Internet thereby making it available to any place that has network connectivity.

In this new networking space, third parties provide APIs that abstract away the details surrounding the implementation, maintenance, and usage of a network connection between programs. As a result, the effort and detailed knowledge about computer networks required to implement and deploy a network-based application is rapidly decreasing. Consequently, more people are interested in developing network-based applications, using these APIs. The result is an increased interest in computer networking but not at the low-level detail that the previous version of our course provided. Collectively, these observations provided the impetus to consider a major revision, akin to developing a new course, and repositioning it.

Although we had a number of goals in mind when revising the course, the key ones with respect to this paper are:

- Employ active pedagogical techniques to improve student learning and produce more expert-like thinking.
- Ensure that assignments had real-world components.
- Create a course more relevant to the broader computer science population.

To provide focus and guidance to this effort we asked a very simple question: "If students take only one computer networking course, what do we want them to learn?" In answering this question, we followed the best practices approach of developing a set of course-level learning goals, or themes, to provide the overall framework for the course content [8]. We then supplemented those with finer-grained goals to inform the detailed development of the course content. The resulting five overall course themes were:

1. Achieving data privacy and isolation
2. Dealing with out-of-order or lost data
3. Maximizing performance via various techniques and approaches
4. Naming and locating entities
5. Using layering and abstraction to build complex systems that can be reasoned about

Although many of these themes were covered in the existing course, they were met by focusing on the internal layers of the TCP/IP protocol stack. Unfortunately, these layers are typically implemented within the operating system and are for all practical purposes inaccessible to the students. The result is a disconnect between the ideas being taught and what we actually expose the students to through exercises and activities. To address this concern we decided to "push" the illustration of as many of these ideas and concepts as possible to the application area, thereby giving students opportunities to explore these themes in a more active way. A simplified version of the course syllabus is shown in Table 1.

## 2.2 What We Measured

One of the challenges when working with new or different pedagogical approaches is evaluating their success. One dimension of success is standard measures of academic achievement on formal assessment instruments such as final exams. Since this isn't a completely new course, we were able to reuse exam questions or produce isomorphic exam questions to evaluate common material between the new and old courses.

A second dimension is more attitudinal in nature where we are interested in evaluating how successful we were at improving

| **Basic Tools** |
|---|
| Protocol state machines, networking APIs |
| Measurement and analysis, bandwidth, latency, throughput, jitter |
| **Application Protocol Case Studies** |
| Client Server, P2P |
| HTTP, SMTP, FTP, POP |
| **Reliable Data Delivery and Performance** |
| Sequence numbers, timeouts |
| Lost data handling, error correction |
| **Naming, Routing, Network Organization** |
| DNS, DHCP, IP Addressing |
| Subnets, Autonomous systems and routing |
| **Privacy and Security** |
| Public Private key encryption |
| Data security and verification |

**Table 1: Revised Networking Syllabus**

student engagement and their confidence with respect to the learning goals. To that end, we did voluntary pre- and post-surveys that focused on the efficacy of our engagement strategies to provide feedback for future course modifications. For example we asked students to rank assignments with respect to both the level of engagement and how useful they were at helping them to learn the material. The effectiveness of the tutorials and the usefulness of the textbook would also fall into this category. On a third dimension we also surveyed students about their confidence with respect to performing certain tasks. For example, we asked about how comfortable they were in using networking APIs, and in working with and developing certain types of applications.

## 2.3 Student Engagement

Using the revised syllabus and learning goals we attacked the problem of increasing student engagement in the classroom. Based on success in other domains, we decided to adopt the pedagogical approach often referred to as a flipped classroom [2]. With the flipped classroom, students acquire basic foundational knowledge outside the class through assigned readings, videos, or a combination of both. A "lecture" then consists of student-centered active learning activities. For example, in an introductory programming course, students might practice producing flowcharts, writing fragments of code, or solving problems that other students have had trouble with in the past. By doing these in the class, students can get immediate help from either the instructor or their peers.

Most of the work with the flipped classroom has concentrated on lower-level courses where the focus is typically on *how* to do something. In upper-level courses the material tends to be more abstract and conceptual, which can present challenges for activity design; therefore, a flipped classroom in an upper-level computer science course is uncommon. In our revised course, a typical class consists of one or two activities combined with class-wide discussions and mini-lectures to bridge and consolidate course topics. A typical activity consists of a problem introduction, the active component (when students work on the problem, usually in small groups to take advantage of peer instruction), and a group discussion to review the issues and discuss what was discovered and its relationship to other course topics. During the active component the instructional team circulates throughout the class to keep students on task and to provide guidance and insight if they are struggling. During the first offering of the course, 36 activities were used. This is nearly two activities per lecture after accounting for five quizzes and other in-class administrative time.

The activities ranged in length from 10 minutes for a simple bandwidth calculation, to a whole class for an invention activity focusing on strategies for data retransmission when creating a reliable delivery channel out of an unreliable channel. Just as the activities vary in length, they also vary in approach. For example when learning about HTTP and FTP, students emulate clients by using telnet and following a prescribed set of instructions on what to do and what to look for. When learning about network organization, they use `traceroute` and `ping` to map out parts of the Internet. When learning about privacy and security, they have to invent a way of exchanging a message using a box, locks, and keys that mimic what happens in an encryption system. To take advantage of peer instruction all of the activities have a group component. On average a given lecture now comprises about 20 minutes of "lecturing" and 60 minutes of student-focused activities.

Yet another way to improve student engagement is to remove distractions, of which laptops can be a major one. To address that, we've tried to incorporate and encourage laptop usage into many of the activities, and created "laptop-free zones" in the classroom for those times when laptops aren't needed.

## 3. CPSC 259: DATA STRUCTURES AND ALGORITHMS IN C

CPSC 259 is a core course for Electrical Engineering students who are *not* in the Computer Engineering option. The majority of our students are in the power systems stream, with smaller numbers in nanotechnology or biomedical engineering. It is a course in C programming, basic algorithms, and data structures—with a small Matlab component. Most of the students do not intend to take further computing courses.

### 3.1 Course Content and Structure

In the prerequisite course, which also uses C, a flipped classroom was used. The class used short online screencasts (built using Camtasia) with voiceovers and interaction (e.g., simulations) for the "lecture" content. Lecture time was then used for problem solving, with students handing in their work at the end of class for participation points. For homework (ungraded), students were given sample programming assignments with full solutions. The homework was designed to prepare them for their weekly programming test. Specifically, they had to take an in-lab programming test, isomorphic to the homework, to demonstrate mastery of the programming concepts. Students who did not do the homework would be very unlikely to complete the lab test successfully. Furthermore, plagiarism was greatly reduced.

In CPSC 259, we are using lectures with PowerPoint slides; Microsoft Visual Studio's C/C++ compiler for in-class demos; and Blackboard's "Connect" course management system to host a large number of course files, including lecture notes (both fill-in-the-blank and (later) annotated slides), copies of programs used in class, and lots of sample exam questions with full solutions. And, of course, we created detailed learning goals for the course, along with a set of programming assignments to support those learning goals.

A key focus of our course was how to engage the students, both in the lectures and in the labs. During the inaugural offering, we had demos in the class, and lots of fill-in-the-blank slides, but we did not use clickers or outside-of-the-class participatory activities. This changed during the second offering.

For the labs, we adopted a *pair programming* model [10] for a number of reasons. First, we know that many students in the course are not really "into" programming, and it made sense to have them work in pairs, so that partners could help each other using peer instruction. Second, research has shown that pair programming is very effective for learning programming. It has been successfully applied in many introductory and sophomore computer courses [1]. Third, students get the opportunity to gain team-building skills, which is a useful asset in an engineering career. Fourth, large class sizes (e.g., 200) make it difficult for TAs and the instructor to help students individually. Fifth, marking is much more manageable with half as many assignments to assess. We gave each pair of students detailed feedback using a marking rubric. Budget considerations would prohibit the detailed marking of 200 students' assignments.

Furthermore, our goals included getting students to *read* well-written, robust code based on industry best-practices. Students provided additional functionality in the code by filling in missing pieces and creating new functions. We had five bi-weekly assignments, divided into two components, each with multiple deliverables. There was an in-lab component during week 1 (much of which would have been checked off during their initial lab), and a take-home component during weeks 1 and 2. The in-lab TA support and TA office hours gave students many opportunities to get help.

### 3.2 What We Measured

Since this is a new course, we decided to implement best practices right from the start by using learning goals, assessment, pair programming, and classroom engagement.

Every unit of the course had a full set of learning goals. We told the students that these learning goals were a useful checklist for studying for exams. Indeed, some of them made good exam questions on their own.

Our first assessment instrument was a diagnostic test. Specifically, we administered a 30-minute pre-test on the first day of class to all students to determine their strengths and weaknesses. The pre-test was not for marks, but was based on the prerequisite course: APSC 160, which uses C. We used feedback from the test to help tweak in-class content in the first few lectures, and to develop some programming assignment questions for the labs. In particular, we chose questions that students had the most difficulty with, and we worked them into clicker questions or a deliverable for Lab 1. For example, only 22.8% of the class got the following question correct on the pre-test; so we addressed it with an isomorphic clicker question during a lecture, followed by peer discussion.

```
Q5.  Consider the following poorly indented
code segment:
     int r;
     int a = -5;
     int b = 6;
     if ( a < 0  ||  b > 0 )
             r = 1;
     else
             r = 2;
             a = 0;
```

```
What are the values of a, b, and r after
this code segment has executed?
```

We also administered voluntary beginning-of-term and end-of-term surveys to capture student attitudes and opinions, especially to reveal what worked, and what didn't. We are particularly interested in seeing how students responded to pair programming. Student attitudes are important because research shows that well-motivated students tend to learn a subject better [4]. Also, we hope that students' *confidence* in programming will improve. Confidence is related to—but different from—their actual scores.

Finally, and most importantly, we are measuring and analyzing midterm exam, in-lab assignment, take-home assignment, and final exam scores. This will allow us to compare the inaugural version with the second (current) version which has more interactive and engaging content.

## 3.3 Student Engagement

In the second offering, to free up lecture time and give the students more practice, we decided to promote more interactive activities inside and outside the classroom. For example, we are using clickers with peer instruction in the classroom [2],[6]; and for additional participation points, we have moderately challenging online tests administered via our course management system. Students can take the tests multiple times; but the software only counts their last attempt at a given test. Furthermore, we have developed a tutorial and simulation about data, data types, pointers, addresses, and structures. For example, students can see what the different data types (representations) look like for the same 32 bits in memory; they can assign and play around with addresses and pointers; and so on.

For the labs, students were required to read the lab in its entirety before coming to the lab. This was to maximize their productivity, and not to slow down their partner. The pair programming model called for students changing positions (one on the keyboard for 15-20 minutes while the other watched for errors, helped, and offered feedback; and then the roles reversed). Apart from the deliverables, students were given both a pair programming and participation grade for each lab; thus, attendance at the labs was usually excellent. Both students in a pair were required to answer TA questions. We provided generous in-lab (e.g., 3 TAs for about 30 students) and out-of-lab TA assistance (e.g., 8 hours per week).

In the second offering, because students were relatively weak in debugging C code, we forced students to actually use a debugger from Lab 1 on, and to demonstrate it to the TAs during the lab. Similar to the first offering, TAs asked questions to both partners about various snippets of code, and to show the TA that they could use the debugger properly. We had checkpoints throughout the lab time so that students frequently interacted with a TA. All of these activities, checkpoints, and deliverables kept students busy during the lab time.

## 4. SUCCESSES AND FAILURES

A full formal analysis of student performance has not yet been done for either course. However, with respect to the revised networking course, preliminary analysis suggests that students performed as well as—if not better (marginally)—on the formal assessments. On the attitudinal side, students appear to be much more confident in their abilities even when their exposure to material relevant to performing a specified task was much less than in the original version of the course. We conjecture that the constant barrage of new activities in the flipped classroom better prepares students for problem solving in unfamiliar areas.

It was noted that flipped classrooms for upper-level courses were not the norm, but because of what was observed in the networking course, these results suggest that a flipped classroom model may be beneficial for other upper-level courses.

More frequent testing in the form of quizzes also seems to bear results because: (a) students claim that it keeps them on top of their work, and (b) the workload for both the instructor and teaching assistants becomes more uniform throughout the term. Educational research states that frequent and cumulative testing is highly effective [7], even though it is not necessarily liked by students. We need to determine the optimal balance between frequency and length of tests, and whether they should be cumulative or not. We are exploring this for several courses.

With respect to what worked and what didn't, one of the most important things to do is to keep a detailed activity and reflection log. In this log record what was done in the lecture (including the time and duration of activities) along with post-lecture reflections about the class and a sense of how the activities were received. After the lecture, we record thoughts about what worked, what didn't, why something didn't work, and any suggestions for changes. Such information can be used for guiding subsequent changes to the courses, or as a reference for someone else delivering the course, or when combined with formal assessments, it is a way of evaluating which activities were the most successful with respect to achieving the desired learning goals.

With in-class activities, we have to be prepared for an activity that goes too slowly or too fast. We encountered this frequently in our networking, data structures, and other courses—especially as more in-class activities are included. The instructor has—to some degree—given up control of the classroom, and has to be prepared to use a dynamic approach to teaching (i.e., "winging it"). This is not necessarily a bad thing because students will raise topics and issues that they are interested in. Basically, when they are curious about something they are receptive to learning. The challenge is making appropriate connections to the desired learning goals—and still fitting in the rest of the curriculum. A perfect example of this revolved around student questions concerning the relationship between multicasting and streaming video. This resulted in a discussion on multicast, Ethernet, MAC addresses, switches, hubs, and even how a home router works, which was all part of the course—even though those topics would have been covered later.

What about attendance? During the first offering of the data structures course a sufficiently large number of students sometimes didn't attend class. At the time of writing, attendance and as well as attentiveness appears to be up for the second offering. We conjecture that this is at least partly due to the use of active learning (e.g., clickers with peer instruction). Many of our clicker questions are not answered correctly at first by a sufficient number of students, so we often go to a round of discussion, whereby students try to convince their peers why their own answer is correct, but their peers' is wrong. A downside to this is that it can be time-consuming; however, it can be argued that most learning takes place *outside* of class, and therefore using this extra time shouldn't be a constraint overall.

One observation about the online tests mentioned above is that 25-35% of the class is not even trying most of the quizzes, even though there are participation points. We conjecture that the number of participation marks is not sufficient to motivate some students. We can tweak this parameter and see what effect more marks would have in future offerings. It is a delicate balance between giving enough marks to motivate students, but not so many that students resort to plagiarism.

Most students responded positively to pair programming in the first offering of CPSC 259; however, we are also aware of a number of students that did not do well. For example, we had a fair bit of plagiarism despite partnerships, lots of extra help, and new labs. Although we assumed that there would be a reasonable division of work between partners this did not always happen (e.g., 0% contribution). Consequently, we now provide explicit guidelines as to how partnerships should work, and what the warning signs are that a partnership isn't working out. We are collecting more data on contribution levels, and are considering alternative approaches, such as short, in-lab programming tests (for each student) to encourage them to actually do the work.

## 5. APPLYING OUR RESULTS

We have greatly improved the quality of the survey questions we've given compared to those of just a few years ago, in our classes. We created some beginning-of-term and end-of-term survey questions that ask for student opinions about very specific criteria. In particular, we have introduced confidence questions about specific tasks to reduce the misinterpretation of a question, and to gather more targeted data. For example, instead of asking students whether or not they can write a 5-page program in C, we might ask them, "How confident are you that you can write a 5-page program in C that uses pointers, and has functions that take pointers as parameters?" Based on the answers to the latter type of question, and combined with questions that rank the effectiveness/usefulness of the course bulletin board, clickers, pre-readings, in-class exercises, pair programming, etc., we can get a better picture of which changes are improving student outcomes.

A lot of the focus on the two courses described in this paper is on how to improve student engagement. As part of our evaluation, we informally observed attendance and attentiveness levels in these courses; however, these are very imprecise observations. To that end, we started using a classroom observation protocol (COP) to determine which elements of the lectures are most engaging, and which are not. Specifically, another instructor or TA observes the class over several lectures. With the COP, the level of student engagement, coupled with the current lecture activity, is recorded every two minutes. As a result, we can make changes to the course, re-do the COP, and evaluate the effectiveness of these changes with respect to student engagement. Additionally, the COP allows us to see where time is "wasted" during a lecture, so that we can recover it. Every delay or distraction has a magnifying effect on disengagement with the rest of the class.

Based upon our results in CPSC 259 and other studies, clickers are one way to engage students. An instructor needs to decide whether or not clicker questions should count for participation points only, for getting the right answer, both, or none. An instructor also needs to decide whether or not to distribute the clicker questions to students for studying purposes, and risk them being made available to students who have yet to take the course. It is important for students to make mistakes in order to learn from them, and to encourage reflection and peer discussion.

Earlier we said that we wanted to make CPSC 317 a course that appeals to more just a systems-oriented crowd. To that end, enrolment in the second offering of the revised course has resulted in a doubling of the number of students. This increase cannot be attributed to program enrolment increases alone.

## 6. FUTURE WORK / CONCLUSIONS

We have two very different courses with different audiences, yet share similar goals with respect to improving student engagement. In this paper, we outlined several different strategies to achieve this. Clearly, it takes substantial effort to modify an existing course or to develop a new course. We still need to do a better job of measuring unproductive lecture time in our courses, and adapting accordingly.

A final thought: With the introduction of Massive Open Online Courses (MOOCs), how can we leverage this new online content to improve engagement in our classrooms? For example, could we use MOOCs to offload some of the lecture content, to more productively engage our students during class?

## Acknowledgements

We would like to gratefully acknowledge the contributions of our colleagues: Bill Aiello, Norm Hutchinson, Mahdi Tayarani Najaran, Jonatan Schroeder, Chris Thompson, and Alan Wagner—as well as the support from the *Carl Wieman Science Education Initiative* at the University of British Columbia.

## 7. REFERENCES

[1] Braught, G., Wahls, T., and Eby, L.M. (2011) "The Case for Pair Programming in the Computer Science Classroom", *ACM Trans. On Computing Education*, 11(1).

[2] Crouch, C., Watkins, J., Fagen, A., and Mazur, E. (2007). "Peer Instruction: Engaging Students One-on-One, All at Once". *Research Based Reform of University Physics*, 1(1).

[3] Gibbs, G. (1981) "Twenty terrible reasons for lecturing". Retrieved on April 11, 2013 from www.brookes.ac.uk/services/ocsld/resources/20reasons.html

[4] Pintrich, P.R. (2003)"A Motivational Science Perspective on the Role of Student Motivation in Learning and Teaching Contexts". *J. Educational Psychology*, 95(4), pp. 667-686.

[5] Prince, M. (2004) "Does Active Learning Work? A Review of the Research". *J. Engr. Education*, 93(3), pp. 223-231.

[6] Ribbens, E. (2007). "Why I Like Clicker Personal Response Systems". *Journal of College Science Teaching*, November 2007, pp. 60+.

[7] Roediger III, H., Agarwal, P., Kang, S., and Marsh, E. (2010) "Benefits of Testing Memory: Best Practices and Boundary Conditions", in Davies, G. and Wright, D. (eds.) *Current Issues in Applied Memory Research*, Brighton, UK: Psychology Press.

[8] Simon, B. and Hanks, B. (2007). "What Value are Course-Specific Learning Goals?" *Proc. ICER'07*, pp. 73-85.

[9] Wieman, C. (2008). "How People Learn—Implications for Teachers and Students". Retrieved on October 7, 2009 from www.cwsei.ubc.ca.

[10] Williams, L. and Kessler, R. (2002). *Pair Programming Illuminated*. Addison-Wesley.