Professor of Teaching Promotion Portfolio: Educational Leadership, Teaching, Curriculum Development, and Service Statement

Steven Wolfman
Department of Computer Science
UBC

September 11, 2013

Contents

1 Introduction

This statement describes my teaching, educational leadership, curriculum development and pedagogical innovation, and service in support of my promotion to the Professor of Teaching. I organized the document according to the Guidelines for Promotion to Professor of Teaching[1]. I include the Guideline's four recommended sections below and reference a set of linked appendices as relevant. I focus generally on contributions that have occurred since my promotion to Senior Instructor on 1 July 2009.

Each section begins with my summary of the Guideline's suggestions for its contents. Where the Guideline indicates multiple locations for a single type of contribution, I briefly indicate which section I chose for the contributions.

Finally, some notes on terminology will be helpful throughout.

1.1 Useful Terms and Definitions

- The word "instructor" means "the instructor-of-record for a course", *not* the rank of Instructor at UBC.
- Several frequent abbreviations merit an early introduction: CS (Computer Science), CPSC (course code for the UBC CS department), TA (teaching assistant, where UTA and GTA are undergraduate and graduate TAs), and URA (undergraduate research assistant).
- For brevity, I will later refer to the following courses by number:

  CPSC 101: Connecting with Computer Science Primarily a non-majors introductory course.

  CPSC 110: Computation, Programs, and Programming Introductory course, a co-requisite for (must be taken before or concurrently with) CPSC 121. Replaced our old introductory course in September 2010. Required for majors.

  CPSC 121: Models of Computation Introduction to discrete mathematics and hardware. First course of the Foundations of Computing sequence and also of the Computing Systems sequence. Required for majors.

[1] See: http://www.hr.ubc.ca/faculty-relations/files/Guidelines-for-Promotion-to-Professor-of-Teaching.pdf.

CPSC 221: Basic Algorithms and Data Structures Introduction to algorithms, data structures, algorithm analysis, and counting. Second course of the Foundations of Computing sequence. Required for majors.

CPSC 311: Definition of Programming Languages Exploration of key programming language building blocks, primarily through construction of successive interpreters. Elective for majors (except for a few specialized streams).

CPSC 313: Computer Hardware and Operating Systems Continued exploration of the system- and hardware-level features of computers that enable effective and efficient program execution. Third course in the Computing Systems sequence (and last one required). Required for majors.

CPSC 320: Intermediate Algorithm Design and Analysis Exploration of families of algorithms and their application, with advanced algorithm analysis techniques. Third course in the Foundations of Computing Sequence (and last one required). Required for majors.

CPSC 448/CPSC 449 Generic course codes for directed studies/thesis courses for undergraduates.

CPSC 490 Generic course code for student-directed seminars, courses proposed by and for undergraduate students. Each course has a faculty sponsor to mentor the proposer and assign official grades.

2 Educational Leadership

This section focuses on formal leadership roles and promotion of teaching and learning where I did not necessarily develop new pedagogy and curriculum. I have excluded the Guideline's suggested "teaching, mentorship and inspiration of colleagues", putting it instead in Teaching, where the Guideline includes the element "participation in the pedagogical training of other faculty and graduate students."

2.1 Leadership in the Int'l CS Education Community

One of my professional goals has been to foster and shape the CS education community, particularly through SIGCSE. SIGCSE is the flagship CS education conference[2]: the Technical Symposium on Computer Science Education, named for the "Special Interest Group" that runs it. It's an annual conference with roughly 1200 attendees, 100 papers, and 3–4 days of content in about 7 parallel tracks. The work described in this section has been in collaboration with many others in the CS education community.

Since my first SIGCSE in 2002, I have benefited from mentorship, training, and networking within the community. I volunteered for years as a student, led the volunteer program in 2006, organized the 30+ workshops in 2008, led the entire technical program in 2009, and the whole symposium in 2010. After a few years' "vacation", I am back leading the volunteer program and as Associate Program Chair.

My most significant contributions to SIGCSE were in 2009 and 2010. The appendix On SIGCSE Leadership includes links to the websites and programs for these two conferences.

As Program Chair for SIGCSE 2009, my co-chair Gary Lewandowski and I focused on improving reviewing, responding to growing discontent from authors. SIGCSE solicits 6 reviews per paper but only 3 per reviewer, which means recruiting roughly 800 reviewers. Gary and I wanted to maintain this odd arrangement as a service to the community, particularly reviewers from teaching-oriented institutions where reviewing is a primary scholarly activity. However, the arrangement produces reviews that are often ill-calibrated.

[2]CS *conferences* not *journals* are the primary venues for dissemination of high-quality work. For one view on this, see the Computing Research Association memo Evaluating Computer Scientists and Engineers For Promotion and Tenure (`http://cra.org/resources/bp-view/evaluating_computer_scientists_and_engineers_for_promotion_and_tenure/`).

We designed a system to balance high-quality, transparent, scalable reviewing with the community's unique requirements. The Program Chairs recruit roughly ten Associate Program Chairs (APCs), each expert in some commonly published areas. Each APC writes meta-reviews for about 30 papers that summarize key points from the reviews and provide a recommendation to accept or reject along with clear, concise reasoning. APCs' place reviews in a larger context, focusing on papers with inconsistent or weakly justified reviews. The system also allows Program Chairs to support work in burgeoning and difficult-to-assess areas, e.g., Gary's and my choice to emphasize parallelism and CS education research.

The APC system was a success in its first year, judging by positive evaluation results and a lack of complaints at the business meeting (a rare event!). The new system has since become standard; indeed, I've rejoined as an APC for SIGCSE 2014.

I also provided a detailed timeline of tasks—with a library of resources such as spreadsheets, database queries, and template e-mail messages—to subsequent years' Program Chairs. Information from my timeline continues to be used today.

Gary and I collaborated again as General Chairs (AKA Symposium Chairs) of SIGCSE 2010. We again highlighted parallelism and CS education research by recruiting keynote speakers on each topic. However, most of a SIGCSE General Chair's effort is unglamorous. Almost all conference administration devolves to us, including scheduling dozens of events and scoping, soliciting, and arbitrating bids on space, A/V, wireless, and catering. In 2010's economic climate, we also struggled to keep registration fees low for our often grant-poor attendees. We held to our budget and increased fees by only \$10 to \$200, *much* less than 2010 CS research conference fees, like \$560–\$760 for CHI, SPLASH (OOPSLA), and VLDB, and rather less than the \$250 fee in 2013.[3]

In 2009 and 2010, I also helped connect the BC computing education community to SIGCSE, consulting with the Western Canadian Conference on Computing Education (WCCCE) Chairs as they obtained "in-cooperation" status with SIGCSE and expanding advertising of WCCCE in particular and of regional CS education conferences in general.

More recently:

- I led a team of faculty from 7 institutions reporting on the roles of teaching-oriented faculty at research institutions and advocating for

[3]All amounts are in US dollars.

key practices to improve their contribution. Our report was published in Communications of the ACM (54(11):35–37, 2011)[4], the main computing professional organization's magazine, sent monthly to all members.

- I acted as Student Volunteer coordinator for SIGCSE 2013 and SIGCSE 2014, creating opportunities for the roughly 100 volunteers to network with each other and SIGCSE "bigwigs", in response to feedback from previous surveys of student volunteers.

2.2 Bachelor of Computer Science Program Director

In my first year at UBC, I taught the "BCS" (Bachelor of Computer Science) section of our introductory course. The roughly 15-person class was tiny compared to my other sections but, as I told the 2013 BCS cohort at their orientation, that class was also louder, more demanding, more intense, more collegial, and more exciting than the huge general sections. In the years since then, I've taught the BCS group and worked with BCS students as often as I could. When I got the chance, stepping up to be BCS Director was a natural move!

I directed the BCS program from 2011–2012 and am resuming the role now after a sabbatical hiatus. The program serves students with a previous Bachelor's degree in another field. It has expanded over four years from admitting 30 students annually to 100. As BCS Director, I am working to reinforce the intense, positive culture that drew me to the program. The work described in this section is in collaboration with Giuliana Villegas, BCS staff lead, and other faculty, students, and TAs involved in the program.

My largest project for BCS is an interactive digital media program entering its second year, a collaboration with Kimberly Voll at the Centre for Digital Media (a UBC-, SFU-, BCIT-, and Emily Carr-sponsored institute). CPSC has no plans to offer digital media coursework, but BCS students frequently express interest in the area. I also observed at a high school career fair that most of the students believe CS *is* digital media. This program fills that gap.

In the 6-credit program, BCS students gain hands-on experience building industry-sponsored interactive media projects with large, multi-disciplinary teams. So far, students and their mentors stress the benefits of the program in terms of technical skills (e.g., web development and version control) and

[4]See: `http://cacm.acm.org/magazines/2011/11/138214-teaching-oriented-faculty-at-research-universities/abstract`.

technical communication with non-technical audiences, as well as exposure to project workflows specific to interactive digital media. (The appendix On the BCS/CDM Collaboration links to the two students' projects.) In the long run, we hope to provide valuable experience to BCS students, create showcase materials for the BCS program and computing education in general, and improve ties with the Centre for Digital Media.

I have also continued the basic services of the BCS leadership, including adding weekly office hours in the BCS room to simplify advising appointments and foster community, improving the application process for international students by gaining Faculty support for an English Exemption Test, supporting a student-organized BCS Club, and consulting with students and the steering committee to understand future directions for the program. A few necessary efforts have had negative results, including: exploring collaborative advertising with aligned UBC programs, finding alternatives to the "Technical Writing" requirement (oft-maligned by BCS students), and addressing a requirement BCS students take an extra course (versus B.Sc. students) for financial aid eligibility.

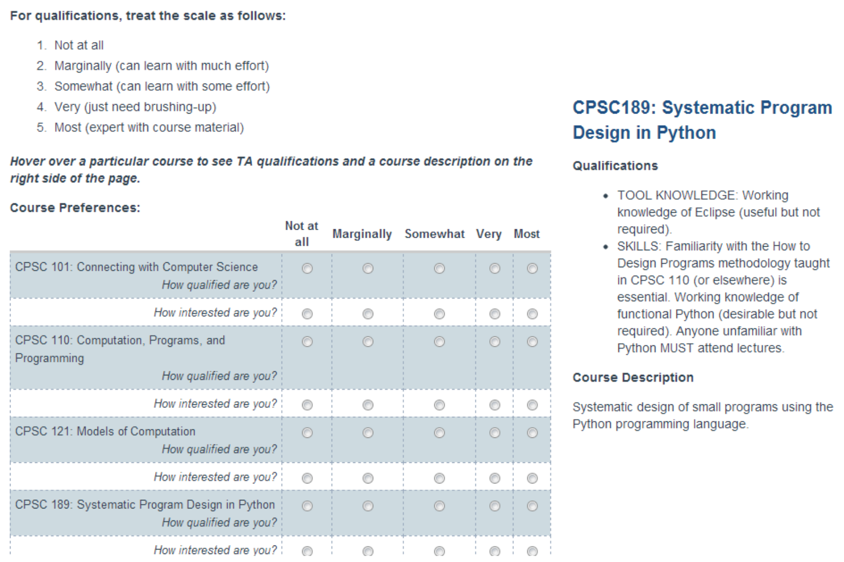
2.3 TA Assignment Coordinator

I was TA assignment coordinator for the CS department from 2010–2012 and am presently resuming the role after a sabbatical hiatus. I took the role as a natural extension of my interest in effective Teaching Staff Management. The coordinator manages the assignment process for undergraduate courses including determining the number of available positions, advertising positions, accepting applications, vetting candidates, making offers, managing changes, and collecting feedback from TAs and instructors on the assignments. In a large department like ours, this is an intense workload—in the fall term of 2013, for example, we eventually hired 68 graduate and 97 undergraduate TAs. The work described in this section has been in collaboration with staff, students, and faculty involved in TA assignment and Renée Stephen, the department's webmaster.

I made important improvements to the "matching" process that associates TAs with courses. In consultation with stakeholders, I redesigned the TA applications for undergraduate and graduate TAs (UTAs and GTAs). The new system increases the information available to the TA assignment team, clarifies the application process for potential TAs, reduces differences in workflow between UTA and GTA assignment, and simplifies processing of application data.

For example, the application (shown below) makes the course content and

TA qualifications clear to TAs. The department already allowed instructors to update course information via editing web account profiles; we added TA qualifications to this area and seeded with qualifications adapted from another committee's survey of instructors.



For qualifications, treat the scale as follows:

1. Not at all
2. Marginally (can learn with much effort)
3. Somewhat (can learn with some effort)
4. Very (just need brushing-up)
5. Most (expert with course material)

Hover over a particular course to see TA qualifications and a course description on the right side of the page.

Course Preferences:

	Not at all	Marginally	Somewhat	Very	Most
CPSC 101: Connecting with Computer Science *How qualified are you?*	○	○	○	○	○
How interested are you?	○	○	○	○	○
CPSC 110: Computation, Programs, and Programming *How qualified are you?*	○	○	○	○	○
How interested are you?	○	○	○	○	○
CPSC 121: Models of Computation *How qualified are you?*	○	○	○	○	○
How interested are you?	○	○	○	○	○
CPSC 189: Systematic Program Design in Python *How qualified are you?*	○	○	○	○	○
How interested are you?	○	○	○	○	○

CPSC189: Systematic Program Design in Python

Qualifications

- TOOL KNOWLEDGE: Working knowledge of Eclipse (useful but not required).
- SKILLS: Familiarity with the How to Design Programs methodology taught in CPSC 110 (or elsewhere) is essential. Working knowledge of functional Python (desirable but not required). Anyone unfamiliar with Python MUST attend lectures.

Course Description

Systematic design of small programs using the Python programming language.

Figure 1: A screenshot of part of the GTA application illustrates the extra information available to students. With nothing but the course title before, grads would likely have no idea that CPSC 189 requires knowledge of the How to Design Programs approach or the Eclipse development environment.

The new application pulls course descriptions and qualifications from this site and exposes it in the application form where applicants self-rate their qualifications and interest in each course. (Applicants previously saw no qualification information and indicated only rough interest levels—UTAs by coarse year-level ratings and GTAs by listing a few favorite courses.) Applicants also provide a detailed justification for their "favorite" courses.

The information available from this application has radically altered the TA assignment process. Applicant self-ratings quickly enable initial assignments that are likely to be satisfactory and successful. The TA assignment team still often verify self-assessed qualifications, but this "digging for information" is rarer and more focused, often guided by the "favorites" information.

The workflow's track record has been good so far. Of well over 100 assigned TAs per term, we typically assign fewer than 10 to courses for which they are unenthusiastic (and none to courses for which they are unqualified). In those rare cases, we explain to the TAs and instructors involved why the assignment occurred. Negative feedback from instructors about the assignment seems to have decreased, particularly from courses with unusual TA qualifications.

I've also made many smaller improvements to TAing in the department: establishing a regular request for instructors to encourage students to TA; organizing panels and social lunches for TAs to help establish a teaching community; increasing transparency by reporting details of the assignment process to stakeholders; and so forth.

Before I left the position for my sabbatical year, I created an annotated standard-of-practice document with a substantial archive of related documents, spreadsheets, and scripts. I supported the new TA assignment team through the handoff process with these documents. With a little help from me during sabbatical, they have maintained the new workflow and a high quality of matching between TAs and courses.

3 Teaching

This section focuses on my teaching, including a brief teaching philosophy statement. However, I generally exclude long-term curricular and pedagogical innovations, which are discussed in Curriculum Development and Pedagogical Innovation instead. I have therefore deemphasized the Guideline's suggested elements "information on new courses, pedagogies, and course content" and "development of new and innovative approaches to education".

Student and peer evaluations of my teaching supplied by the UBC CPSC department are available in the Teaching Evaluations appendix.

3.1 Teaching Philosophy

My first task as teacher is to help students find their motivation to learn by making CS exciting and contextualizing it, e.g., through connections to research, everyday problems, and intriguing puzzles. Once motivated, students learn best when they are *active*, *reflective*, and *social*. They *actively* engage: solving problems, discussing ideas, and otherwise exercising their skills. They *reflect* by investigating what and how they have learned and how to learn more effectively. They rely on *social* support (peers, mentors) to facilitate and give context to their learning. I cultivate these traits through pedagogical techniques (such as novel active learning methods for large lectures), technologies (such as new presentation tools for dynamic lectures), and community activism (such as CS student discussion groups).

Examples of how this philosophy expresses itself in my teaching can be found throughout the portfolio. Student evaluations—particularly items about presenting material interestingly and stimulating students to think—and peer evaluations of my teaching provide evidence that I successfully excite students about computing. Contextualizing CS gives a few brief examples of how I connect course material to other domains. Inverting the CPSC 121 Lecture focuses on how I've made student learning in CPSC 121 classes more active while the project I developed for CPSC 221 described in Educational Technology is an example of how I made student learning more active outside the classroom. Pedagogical Training for Colleagues describes some of the work that has stemmed from reflection on my own teaching while the staged milestones in the CPSC 221 project described above are an example of how I encourage students to be reflective learners. Student Mentoring describes some of the work I've done to encourage socialization and mentorship around learning for students; Educational Leadership repeatedly touches on my efforts to encourage socialization and mentorship among students, TAs, and colleagues.

3.2 Student Mentoring

My success as a student came with the support of an amazing group of mentors, faculty, staff, and senior students who gave their time and expertise to help me succeed. This kind of mentorship, always beyond the classroom and often informal, is critical to students' success and their development as professionals. Like my own mentors, I try to form strong relationships with my own students. When I don't believe I'm the best fit as a student's mentor, I also play "matchmaker" with my colleagues, recognizing when others have the technical or soft skills to help a student that I might lack!

Since 2009, I've formally mentored 10 undergraduate students through research assistantships, directed studies, and honours theses; 5 undergraduate Integrated Science students (as their assigned supervisor); helped supervise one Ph.D. student's thesis work; acted as second reader for two Master's student's theses; acted as teaching mentor to a graduate student teaching for the first time; sponsored two student-directed seminars that explore topics unavailable in the curriculum (CS Education and "Big Data"); and secured $4600 of funding for undergraduate researchers, with approximately $3000 additional under review.

Publication is not necessarily the end goal in my mentoring relationships; however, many of my URAs, directed studies, and thesis students have published with me. Tian presented at a UBC research conference; Piam, Lisa, Elizabeth, and Kuba have all presented posters at an annual UBC Carl Wieman Science Education Initiative event; Chris and Elizabeth published papers at SIGCSE; Kuba published a poster and has a paper in submission; Erica took third place at the World ACM Student Research Competition finals; Arianne and I prepared a set of presentations on elections that I've used at many UBC and Vancouver community venues; Kevin and Caroline's projects are both getting live use in industry; and Stephanie plans to publish her thesis. (Most of these appear as publications or annotations in my CV. Sample election materials and Kevin and Caroline's are linked from the appendices.)

In 2011-2012, I also organized a weekly lunch session open to anyone in the department where we "shoot the breeze" about CS topics, usually guided by a paper or brief presentation by someone in the group. Typically these draw a mix of 5–10 undergraduate and graduate students. Students learn CS material and establish social ties that will help them in their studies and careers. For example, a mathematics graduate student found mentors to help her connect her thesis to related CS topics through the lunches.

Finally, I have an "open door" policy with my students and TAs—that is, if my door's open, come in and talk to me about anything (and I try to leave it open as much as possible!)—that has led to a great deal of informal mentoring, especially since I become an official departmental advisor. I keep no statistics on these sessions, but an example of their impact was guiding three of the most dedicated students in my programming languages course into research in the area after they finished.

3.3 Pedagogical Training for Colleagues

Since the second class I taught, when I first had a co-instructor and a large teaching staff, I've tried to maintain a vibrant community of pedagogical practice around me. Early in my faculty career, that meant founding a weekly lunch session for our department's teaching-oriented faculty. Since then, I've continued to encourage and guide my colleagues' pedagogical development through efforts at various scales. Focusing on the time since 2009, this has included:

- co-organizing and participating in weekly CS education reading groups,
- sponsoring a student-directed seminar on CS education research and practice aimed at TAs,
- participating and presenting with the Carl Wieman Science Education Initiative reading group,
- working with UBC's Office of Learning Technology to establish online midterm evaluations for CPSC courses,
- attending and presenting at multiple UBC events on designing learning goals, adapting Just-in-Time Teaching, and using clickers effectively,
- co-founding and participating in the UBC Peer Exchange Network that matched instructors in different science disciplines to trade formative peer teaching evaluations (described in a publication at the International Conference on Improving University Teaching in 2009),
- formally and informally mentoring three new instructors in our department as well as being a panelist at a mentoring workshop for new CS educators at SIGCSE 2011,

- securing funding for and arranging invited talks from CS education researchers,
- giving invited talks on education research at several conferences and at U. Toronto and U. Washington (shown in the appendix On Concept Inventory Development),
- and publishing and presenting peer-reviewed educational research and pedagogical insights in local, regional, and international venues, including much work since 2009 listed in my CV.

Additionally, colleagues in my department have adopted approaches I've designed from each of the recent courses I've taught. In CPSC 121 and CPSC 221, this included adoption of my revised approach to the courses and a full set of curricular materials from lecture slides to weekly TA meeting agendas. In CPSC 320, subsequent terms have adopted my first-week introduction of a running example that connects almost all the later topics in the course (linked from the appendices). In CPSC 101, subsequent instructors adopted my lab updates and roughly a week of lecture materials. In CPSC 311, the only other instructor since I started work on the course based his exams and overall approach on the structure I laid out; I am also giving extensive feedback to the textbook author as he reworks the textbook to a new approach.

3.4 Educational Technology

After my Ph.D. work on pen-based educational presentations, I have continued to innovate with and adopt educational technology. During my first years at UBC, this included such efforts as extending my Ph.D. research to cell phones with a URA, popularizing tablets as teaching tools through a faculty tablet loaner program, and piloting use of clickers.

Focusing on the time since 2009, I co-authored a paper with my colleague Kelly Booth and Ph.D. graduate Joel Lanir on patterns of use of dual-display technology, using a tool Joel developed and I helped prototype. His software has been widely deployed across disciplines at UBC, and I regularly use it in my classes, particularly for guiding students through extended in-class problem sessions.

With several TAs and two URAs, I led development of an unconventional programming project used several times at UBC and once at U. Toronto. The appendix On CPSC 221 includes documents describing the project in detail. Briefly, students receive an automatically personally tailored program with "mystery" data structure implementations and develop experimental methods—test suites and associated documentation—to distinguish the implementations. They submit their performance testing suites to an automatic grading and leaderboard system. Students can earn an A by completing the project's basic goals but can also compete to reach the top of the leaderboard, for fun and a little bonus credit. Our analysis of exam results between terms that did and did not offer the project substantiates clear learning gains. We published this work in a poster at SIGCSE 2013. Equally important, our analysis of the exams and project reports surfaced crucial misconceptions about asymptotic analysis and data structures. These analyses led into the work documented in Foundations of Computing Concept Inventory.

I also continue to develop online quizzes for CPSC 121, participated in UBC's Blackboard pilot for CPSC 101, adopted Piazza for CPSC 121 and tried a competing system developed by UBC students (now retargeted to industry) for CPSC 311, use clickers extensively, develop a variety of web-based materials (with each of my courses having a website and usually a public archive of past websites), develop custom scripts to support various aspects of teaching, and so forth. My reviews for CHI—the premier human-computer interaction conference—were also solicited because of my expertise in educational technology.

3.5 Contextualizing Computer Science

I make regular, broad connections between students' course material and many subfields of CS, other disciplines, and key current events. For example, I augmented CPSC 221 with research papers and a project on Cuckoo Hashing, a surprising recent development in hash tables. I connect CPSC 121 material to democracy and elections by discussing Arrow's Impossibility Theorem (a profound but distressing, Nobel Prize-winning result in economics). In CPSC 320, I tied together almost all the topics in the course with a running example of the stable marriage algorithm, which we connected to Canada's current hospital resident matching system through both lecture and assignments. The appendices include links to sample CPSC 121 election materials and the CPSC 320 example.

In general, each of my classes makes at least one "connection" each week (and usually many more), whether it be to advanced research in an area, to another subfield of CS, or to another discipline.

As part of my efforts to contextualize CS concepts, I keep up on research in human-computer interaction and programming languages by advising student research, reviewing papers, attending reading group meetings, and reading the literature myself. Human-computer interaction was my Ph.D. thesis area and connects to many other areas of CS. I'm familiarizing myself with programming languages research after becoming the primary instructor for our programming languages course in the face of a shortage of faculty expert in the area. (It might even be conceivable by now to say that I have some expertise in the area!)

3.6 Teaching Staff Management

In the first two courses I taught, I managed a total of 20 teaching assistants. Their contribution to the courses was tremendous and has inspired me to strive since then to improve my management of TAs, increasing the TAs' value to the students and the value the TAs' get out of teaching. Back in 2002, that meant publishing a paper at SIGCSE that discussed how to leverage advantages of large TA teams, among other topics. Most recently in 2013, I presented to roughly 100 TAs at our department TA training on "How to Go Above and Beyond", and I kicked off a new collaboration with a URA to identify best practices in course staff meetings.

Throughout this time, I have led management of teaching teams in my large class offerings (CPSC 101, 111, 121, and 221, with about 10-15 TAs each). The duties I take on include: streamlining and improving assignment of TAs to duties through pre-term surveys, focusing staff meeting time through pre-posted agendas, and encouraging TA feedback throughout the term. In large classes, these staff meetings are like teaching an extra seminar alongside the course, with just as much learning going on for the "students" (and for me!) as in the regular course. The appendix on On CPSC 121 includes a sample staff meeting agenda.

My efforts with staff management have had particularly substantial impact in CPSC 121 (described in part in CPSC 121 Curriculum Development), where I also manage long-term continuity of the TA leadership team. I published with Elizabeth Patitsas and Meghan Allen on some of this work at SIGCSE 2011.

4 Curriculum Development and Pedagogical Innovation

This section focuses on *sustained* curriculum development and pedagogical innovation and its broad dissemination. Because all of my publications since 2009 fall under the Guideline's suggested element "contributions to the scholarship of teaching and learning and resulting publications", I refer readers to my CV for a list of these publications.

4.1 CPSC 121 Curriculum Development

CPSC 121: Models of Computation is one of only three first-year CS courses and one of the two required for majors. The course is an unusual mix of theoretical and hardware foundations of computing. When I began teaching it, it was also a course with a reputation as the "black sheep" of the curriculum. The department had been concerned enough to use student focus groups to "debug" the course.

I have been engaged in a long-term effort to renovate this course with research-based educational practices guided by feedback from course staff and students. I've disseminated this work through various publications at SIGCSE, WCCCE, and locally at UBC. The changes have occurred in three major areas:

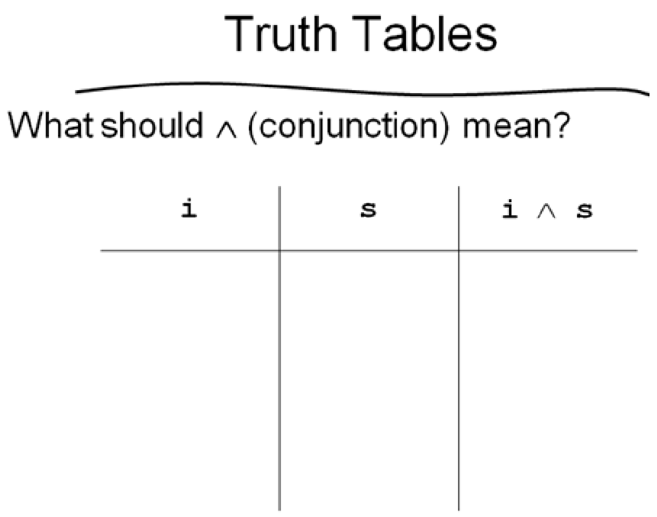
- improving student learning and experience in lecture with Just-in-Time-Teaching [3] and active learning (an "inverted" classroom, although the term didn't exist when this effort started),
- restructuring the course's labs, both its materials (to reduce unnecessary student confusion, connect with the computer model used in CPSC 313, and promote authentic inquiry) and its staffing (designing a TA workflow that incorporates careful preparation and feedback on labs and establishing TA leadership roles to make the workflow sustainable),
- and more recently, adapting to significant changes made to our other required first-year course, CPSC 110: Computation, Programs, and Programming.

The work described in this section has been in collaboration with the course staff of several iterations of CPSC 121, particularly Patrice Belleville and Elizabeth Patitsas.

4.1.1 Inverting the CPSC 121 Lecture

I was inspired to "invert"[5] the CPSC 121 lectures through the Carl Wieman Science Education Initiative and my colleague Paul Carter's experimentation with Just-in-Time Teaching (JiTT). After reviewing my lecture materials, I felt too much in-class time was spent on basic concepts that could as well (or better) be learned *before* class with appropriate resources, particularly in the first half of the term. As a result, class time can focus on critical, high-level concepts and application of concepts to real problems. The appendix On CPSC 121 includes a sample of materials from one JiTT unit.

For example, I used to spend a substantial fraction of a lecture giving the symbols, names, and truth tables for common logical operations. These are concepts firmly in the "knowledge" area of Bloom's Taxonomy, which students are fully capable of learning on their own with some guidance and practice. Unfortunately, students made little use of the textbook, their primary tool for preparation. Why should they when lecture recapitulated the textbook?



Truth Tables

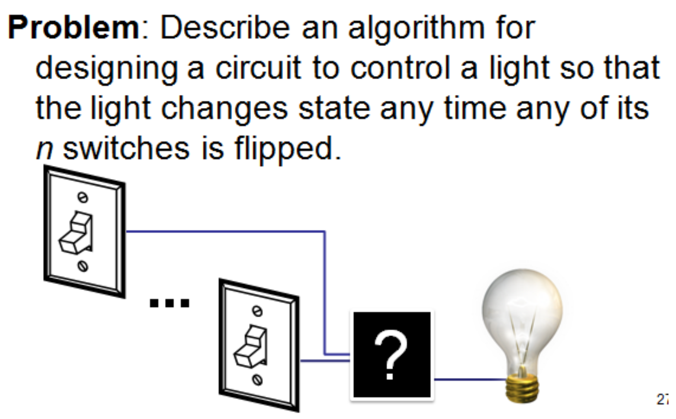
What should $\wedge$ (conjunction) mean?

i	s	i $\wedge$ s

Figure 2: A slide from before converting CPSC 121 to Just-in-Time Teaching (JiTT). A series of slides would introduce truth tables for various operators; I would fill them in as we went.

In a nutshell, an instructor using JiTT tells students to read the textbook

[5] I use "invert" rather than "flip" since I do not use videos for pre-class instruction.



Problem: Describe an algorithm for designing a circuit to control a light so that the light changes state any time any of its *n* switches is flipped.

Figure 3: After converting to JiTT, this lecture focused on developing a solution to this open-ended problem.

and means it. The original JiTT formulation encourages student preparation and has the teacher respond to students' pre-class work on a single broad, open-ended question requiring synthesis of the readings. The instructor scans the responses, develops a gestalt of students' progress, and uses the open-ended question to guide a highly interactive in-class discussion.

Unfortunately, this approach has two major problems. First, as with the common—and amply refuted—wisdom that an experienced instructor can just "look for glazed eyes" and gauge how well students are learning, I suspected I'd have difficulty gauging student progress on low-level learning goals by quickly reviewing open-ended responses to a synthesis question. Second, JiTT simply didn't work well enough in practice at encouraging textbook reading, as the figure below shows.

Therefore, my adaptation of JiTT included additional features to mitigate these problems. In all, the process for a particular unit looks like:

- With the *previous* unit's lecture slides, I distribute *pre-class* learning goals for the current unit, which students achieve outside class. I assign textbook readings or supplementary materials to help them. Pre-class goals are firmly rooted in the "knowledge" and "comprehension" levels of Bloom's taxonomy.

- At about the same time, I post an online quiz for the unit with a section of closed-ended, low-level questions automatically marked for correctness plus one or two high-level synthesis questions marked for

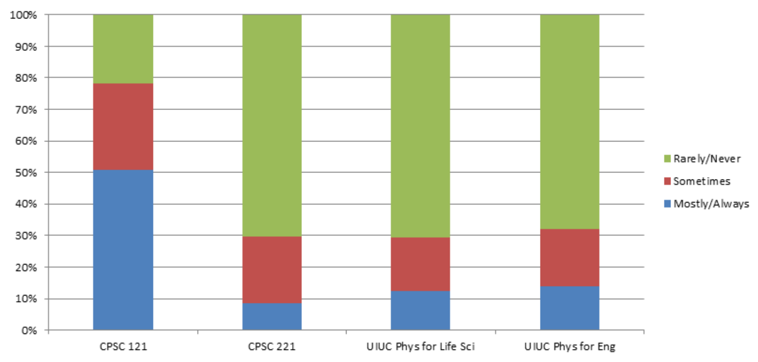


Figure 4: Self-reports of textbook usage across four courses indicate that "normal" courses (the term of CPSC 221 listed) and conventional JiTT courses (two longstanding JiTT courses at UIUC [4]) have trouble encouraging textbook usage. CPSC 121 students make much greater use of their textbooks with JiTT modified to include pre-class learning goals and pre-class quizzes with both closed- and open-ended questions.

completeness only. The quiz has no time limit, and I encourage students to use it to guide their study. The quiz is due just before the lecture unit begins. Students see grades and answers (to the closed-ended questions) shortly after the due time.

- Before lecture, I analyse the closed-ended results, designing mini-lectures and clicker questions to address problems.
- Also before lecture, I skim open-ended responses and summarize for use in guiding interactive lecture inspired by the question.
- The current unit then begins with a review of quiz results and one to three higher-level *in-class* learning goals that we accomplish through interactive, clicker- and discussion-driven lecture.

Online quiz and clicker results from previous terms help me tune and improve the materials each new term. Student feedback has also helped tune the process. For example, in response to educational research literature and student requests, I incorporated additional worked examples into lecture notes, added a "where are we in the big picture" section to each unit, and shifted slide presentation toward sparse and visual information.

Another key to long-term development has been to "trim back" some previously added elements. (Our cross-course studies within CS suggested that by 2010 we had made 121 an extremely high-workload course!) As a result, student workload assessments have returned to reasonable levels.

4.1.2 Restructured Labs

We made radical changes to the labs of the Jan–Apr 2009 CPSC 121 offering, particularly to make inquiry in the labs more authentic: asking students questions to which they did *not* already know the answers.

However, along with general polishing of the new materials, there were two other significant problems. First, the digital logic simulator used in the labs was unwieldy for the bulk of students unfamiliar with the UNIX operating system. Second, the workflow for lab TAs' preparation was unclear and sometimes ineffective.

We addressed the first problem by researching available digital logic simulators, settling on Logisim because it supported interface idioms familiar to most students, gave real-time feedback on changes to the circuits, and made circuit layout changes quick and easy. The new system also works better for in-class demonstration and exploration of circuits. We then switched the old labs to Logisim. We also took the opportunity to shift our old full-computer simulation to a Logisim implementation of the one used in the later CPSC 313 course, giving students more opportunities to revisit the same design. (Although I created an initial draft of the Logisim computer, Patrice Belleville definitely led this implementation effort.) My TA and URA Elizabeth Patitsas documented a large, sustained, positive shift in students' satisfaction with the digital logic software.

In response to the second issue, I worked with Elizabeth and subsequent TAs to create and document a standard workflow for managing the labs and lab TAs. There are now two defined lab TA leadership roles—Manager (day-to-day operations) and Planner (term-to-term revisions)—and documents detailing the purpose, content, and revision history of the labs themselves and of the two positions.

The new workflow for a lab runs from two weeks before the lab starts until a week after it ends, with followup between terms. Two weeks prior to the lab, the Planner reviews a draft lab with the instructor, advising the instructor of any prerequisites expected from lecture. The Manager then runs a prep meeting in which all lab TAs work through the new lab. The Planner incorporates their feedback into the draft lab. A week before lab, the Planner posts the lab to students, getting more feedback from keen students. During the lab week itself, the Manager keeps the TAs posted on any errors or problems. At the following lab prep meeting, the Manager solicits more feedback. At the end of term, the Manager runs a final session to get overall feedback on all labs. The next term's Planner uses all of this plus student survey in her revisions.

This arrangement has worked smoothly for approximately two years. I ensure continuity in lab TA leadership by working with the present Lab Planner and Manager to recruit replacements as they prepare to depart.

4.1.3 Adaptation to CPSC 110

In 2010, the department replaced the first course in the Software Development stream. The new course (CPSC 110) represented a radical departure from the old one but became the new co-requisite to CPSC 121. I have therefore been working to adapt CPSC 121 to CPSC 110.

My experience with CPSC 311, which takes a similar pedagogical and software design approach to CPSC 110, has helped with this adaptation. Additionally, I participated in a brief workshop in 2010 introducing the new course's approach and in the summer of 2012 sat in on the first two-thirds of the course—the most I felt CPSC 121 could rely on from students taking both the courses together.

I have now adapted CPSC 121 substantially to CPSC 110. In some cases, new approaches in CPSC 110 offer serendipitous opportunities we exploit in CPSC 121. For example, CPSC 110 produce conditional guards and hear that different guards can result in equivalent programs. I rebuilt CPSC 121's logical equivalence unit to exploit this point, showing students how CPSC 121 skills can help them understand and modify CPSC 110 programs. I also adapted the course to timing changes in CPSC 110 (like early availability of recursive structures) and high-level design changes (particularly the pattern-driven use of the structure of data to drive the shape of the program). As an example of both, I used CPSC 110's data definition for binary trees to drive our discussion of induction, refocusing on structural induction. I carried this structural focus to many other examples, e.g., recasting weak induction over summation to structural induction with a recursive definition of the Σ operator. I also designed a handout (included in the appendix On CPSC 121) illustrating how the structure of a predicate logic theorem can drive the structure of its proof. Along with these conceptual changes, I've also converted a variety of examples from the old course's programming language (Java) to the new one (Racket).

4.2 Parallelism in CS Education

Since about 2009, inspired by UBC colleagues Mark Greenstreet and Alan Hu and by colleagues in the CS education community, I have engaged in a long-term effort to reintroduce parallelism—the simultaneous use and management of multiple computational resources—into CS education locally and internationally.

As discussed in Leadership in the International CS Education Community, I pushed to incorporate more parallelism into SIGCSE, the premier annual CS education conference. However, my own technical background in parallelism was weak; so, I attended workshops at SIGCSE 2011 and 2012 on incorporating parallelism in data structures courses. I also attended Facebook's "Facebook in Education" workshop at their headquarters in 2011, focusing on its parallelism content.

In collaboration with Alan Hu, I adapted material from Dan Grossman's SIGCSE 2011 parallelism workshop into a three-week unit in the Jan–Apr 2012 term of CPSC 221 (data structures and algorithms). I made two substantial changes from Dan's materials. First, I switched the language of the materials—lecture notes, assignments, sample exam questions, and a 64-page mini-textbook—from Java to C++. This required significant research, writing, and programming effort, such as generating sample programs for the substantial 2011 revision to the C++ standard.[6] Second, I entirely rewrote the lecture materials to take an active learning approach. My new materials have since been reused in CPSC 221. Also, responding to requests for a C++ 'port, Dan incorporated my materials into the repository he maintains. These materials are linked from the appendix On CPSC 221.

4.3 CPSC 221 Curriculum Development

Beyond the parallelism material described above, I have substantially revised the materials in CPSC 221 to emphasize design tradeoffs in implementation of abstract data types, revisions that have been adopted by several other course instructors. I developed a set of new programming projects and adjusted them to include progressive milestones, considerably improving student success in the programming portion of the course. One of these projects is described under Educational Technology.

[6]Although the C++ standard was released in 2011, few courses have begun using it; I believe our CPSC 221 is the first in my department.

4.4 Foundations of Computing Concept Inventory

Much of my curriculum development work has been directed at courses in the core "Foundations of Computing" stream: CPSC 121, CPSC 221, and CPSC 320. I am naturally curious to assess the work's impact on student learning; however, assessing learning gains is a tricky business. As Allison Elliott Tew and Mark Guzdial put it: "[C]omputing lacks valid assessments for pedagogical or research purposes. Without such valid assessments, it is difficult to accurately measure student learning or establish a relationship between the instructional setting and learning outcomes." [5]

Inspired by Allison's work and in collaboration with her and fellow instructors, TAs, and URAs, I am leading a project to develop a concept inventory for the Foundations of Computing sequence. My long-term goal is a sustainable "thermometer" that can track the health of our Foundations of Computing stream in order to guide changes in curriculum and pedagogy.

Our process adapts from those proposed by Adams and Wieman [1] and Almstrum *et al* [2]. In particular, we have so far:

- identified key concepts based on interviews with subject matter experts—including 9 recent instructors of our Foundations of Computing courses—and analysis of student artifacts from the courses—including high-level analysis of over 1500 final exam papers,
- collected information about student misconceptions through deeper analysis of course artifacts—including more than 100 exam papers and project submissions—and 25 one-hour think-aloud interviews with students, and
- formulated multiple choice questions to probe misconceptions of interest.

We have also begun validating some multiple choice questions through think-aloud interviews, ensuring that students choose their answers (correct or otherwise) for the reasons we expect. Approximately 150 students have so far written drafts of our multiple choice assessment, including both pre- and post-tests for one group in CPSC 121 (the first-year Foundations of Computing course).

The appendix On Concept Inventory Development includes a paper detailing this work for five misconceptions related to CPSC 221 and a presentation discussing work-in-progress overall.

4.5 Computing and Democracy

My use of connections between computing and democracy is a good example of how I contextualize computing. I first looked at the topic when British Columbia almost adopted the Single Transferable Vote in 2005. At the time, I worked details of the voting system into course assignments in CPSC 221. Since then, particularly while working with URA Arianne Dee, I've explored the topic much more thoroughly. Arianne and I developed an annotated bibliography and participated in the annual "Workshop on Technology in Elections" in 2011.

I've created approximately 10 hours worth of lecture material (with clicker exercises) targeted at connections between elections and the areas of algorithms and game theory, cryptography, and human-computer interaction plus associated learning goals, assignments, and exam problems. I've used these materials frequently in my own courses and guest lecturing in others' courses as well as in invited presentations to upper-level honours CS students, prospective undergraduates and their parents, and a group of concerned activists at a Vancouver community event. The appendix On Computation and Democracy includes some of these materials.

5 Service

This section focuses on service that is not directly related to teaching and pedagogy or not explicitly described in other sections. Because my CV briefly lists my service contributions, I try here to provide context for the contributions since 2009.

Besides the activities listed above (particularly my work as Bachelor of Computer Science Program Director, TA Assignment Coordinator, and various roles with SIGCSE organization described in Leadership in the International CS Education Community), I've engaged in a variety of service with the CS department, UBC, and the community:

- I frequently lead and participate in activities to build connections between students and faculty (town halls, orientation events, *etc.*), such as leading several hour-long orientation day events to "demystify" university faculty for new students, the most recent in 2013.
- I've served in a variety of departmental and Faculty committees, including the following:
  - The CS department Student Development committee, where I focused on establishing a series of TA panels and lunches to build community among TAs. (Survey results showed that these events were successful in helping junior TAs find mentorship.)
  - The CS department Science Education Initiative committee, where I helped to recruit Fellows to work with the program and organize invited talks on topics involving CS education.
  - The CS department Outreach committee, which I've recently joined with the intention of polishing roughly 12 hours of materials for early K–12 math/CS education I developed and presented at a local elementary school with my colleague (and spouse) Rachel Pottinger.
  - The CS department Merit committee and Undergraduate Student Service's Working Group, where I led no projects but supported the work of the committees
  - The TA Operations Working Group, *ex officio* as TA assignment coordinator.
  - The Faculty of Science Killam Teaching Award selection committee, for which I've observed lectures and reviewed nomination

packets for dozens of faculty across Science disciplines. Work on this committee also improves my own teaching by giving me insight into the techniques used by superb teachers from many different disciplines.

- Along with copious advising as BCS Director, I have also been a drop-in advisor for the general pool of CS undergraduate students and a 1-1 advisor for several Integrated Sciences students interested in CS. I provide career and course selection guidance, check program requirements, suggest extracurricular opportunities, and the like. BCS and Integrated Sciences students require extensive support because of their diverse backgrounds and customized programs. (For example, in one of my BCS office hours, I worked with a BCS student to clear 12 credits of fourth year music research project courses with the Faculty of Science and another trying to decide between upper-level Physics and Economics courses for his program.)
- I reviewed for various CS Education and Human Computer Interaction venues, including serving on the editorial board of the CS Education Journal, devoted to publishing high-quality CS education research.

References

[1] Wendy K. Adams and Carl E. Wieman. Development and validation of instruments to measure learning of expert-like thinking. *International Journal of Science Education*, 33(9):1289–1312, 2011.

[2] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. Concept inventories in computer science for the topic discrete mathematics. *SIGCSE Bull.*, 38(4):132–145, June 2006.

[3] Gregor M. Novak, E. T. Patterson, A. D. Gavrin, and W. Christian. *Just-In-Time-Teaching: Blending Active Learning with Web Technology.* Prentice Hall, 1999.

[4] Timothy Stelzer, Gary Gladding, Jose P. Mestre, and David T. Brookes. Comparing the efficacy of multimedia modules with traditional textbooks for learning introductory physics content. *American Journal of Physics*, 77(2):184–190, 2009.

[5] Allison Elliott Tew and Mark Guzdial. The fcs1: a language independent assessment of cs1 knowledge. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 111–116, New York, NY, USA, 2011. ACM.