Teaching Statement—Phillip T. Conrad

June 30, 2017

This teaching statement is prepared for a career review for promotion to Senior LSOE. The review period is from initial appointment as LPSOE, November 1, 2007, to June 30, 2017; this statement summarizes my teaching contributions during this period.

Teaching Responsibilities

My teaching duties are divided evenly between between the Computer Science department in the College of Engineering (CoE), and the Computing Program (previously called the Computer Science program) of the College of Creative Studies (CCS).

Courses in CoE CS are typically lecture sections of 50 to 150 students, with smaller discussion sections of size 30, and have TA support. Courses for CCS are typically small seminar style courses (10-20 students), and have no TA support.

Learning Philosophy

By calling this my "learning philosophy", rather than my "teaching philosophy", I have already tipped my hand. My learning philosophy is guided by two principles, both of which come from the work of John Biggs, a researcher on effective teaching and learning in higher education[1].

The first is that the most important factor in what a student learns is ***what the student does, not what the teacher does*.** The second is the principle of ***constructive alignment***—which simply put, is the practice of ensuring that all of the components of a course—e.g. learning outcomes, exams, assignments, readings, lectures—are aligned with one another. Both of these principles may seem obvious. However, it is quite common to see situations where they are not applied in undergraduate education. Teaching in the absence of learning is wasted effort. I see my role as teacher as one of **creating favorable conditions** for learning, and **providing useful feedback** on learning.

In addition to constructive alignment, these favorable conditions also include:

- a course structure that **motivates learning** makes it clear to students how the learning objectives of the course align with their personal goals—be those short term goals of earning a particular grade, or long term goals involving intellectual growth or career opportunities.
- a **respectful learning environment**—one in which students of a diverse backgrounds feel respected and welcomed.

A common starting point for undergraduate courses is the construction of a set of lecture notes—and this is often the means by which a course implementation is handed off from one instructor to another.

Biggs' work suggests a different starting point—namely, learning goals. These may be expressed in high-level terms, but become concrete when they are turned into the summative assessments used in the course: exams, and graded assignments in which students demonstrate mastery of the learning goals. The learning goals then suggest the structure of student assignments where the learning takes place, as well as formative assessments to give students and instructors feedback on how they are doing. Finally, the student assignment suggests the selection of readings and content and structure of class activities (including, perhaps, some lecture) that prepare students for the learning.

Note that the readings and lectures are not the most important contributors to the learning itself, but are secondary—they are preparation for where the deep learning takes place, i.e. when the

[1] The theory of "constructive alignment" comes from the work of John Biggs, especially *Teaching for Quality Learning at University*, Buckingham: Open University Press/McGraw Hill, 2011, 4th Edition.

students practices the skill or applies the knowledge gained during the reading or lectures, in order to make it "stick". In the context of Computer Science, these student activities typically consist of programming assignments, mathematical proofs or arguments, or solving a pencil/paper problem of some kind.

This philosophy guides my teaching in both CoE and CCS courses, though the differences between these learning contexts changes how these principles are applied.

In CCS, the smaller class sizes permit learning outcomes to be somewhat personalized and directly observed. CoE courses provide a different context. Typical class sizes for the courses I teach in CoE have grown over the time I've been here by about 50%—from 50-100, to 75-150, and there is the potential for them to grow larger. In these contexts, learning activities must be more standardized to allow course designs to scale.

Nevertheless, in both cases, working towards ever increasing levels of constructive alignment for effective learning in my courses is where I place my primary intellectual effort as an LSOE.

Teaching Contributions

Over the past nine years, my contributions for the College of Engineering CS department have included:

- Developing three new programming courses for the required lower division sequence: CMPSC 8, CMPSC 16, and CMPSC 56. I was the first instructor to teach each of these, and many of the materials I developed have been used by subsequent instructors.
- Significant revisions to all five of the CoE courses I've taught, namely the three above, and two other required lower division courses: CMPSC 32 and CMPSC 40.

For the College of Creative Studies Computer Science Program (renamed to the Computing Program in 2015), my teaching contributions have included:

- Restructuring the Fall introductory quarter for incoming CCS Computing majors, creating distinct roles for the courses numbered 1A and 1L.
- Introducing a research seminar for CCS Computing students that has contributed to a significant increase in the number of CCS Computing students engaging with tenure-track CS faculty in undergraduate research in Computer Science.
- Developing new upper division elective courses in Web Development, Digital Audio Programming, and Computer Science Education that have served both CCS and CoE Computer Science students.

The remainder of this statement will provide more detail about each of the items listed above, then conclude with a summary of the principles that guide my teaching and curriculum development.

Note on Course Evaluations

UCSB policy prohibits the inclusion of full student course evaluations reports in the material sent to external reviewers; instead, candidates are invited to discuss these evaluations in the teaching statement, and I have done so in places where it seemed relevant. For reference, here are the questions asked at UCSB of every student, in every course, every quarter:

(A) Please rate the overall quality of the instructor's teaching independent of the course materials or content

(B) Please rate the overall quality of the course, including its material or content, independent of the instructor's teaching.

The scale for both questions is: (1) Excellent (2) Very Good (3) Good (4) Fair (5) Poor. Narrative comments are also invited, and in some cases, quotes from those are included.

CMPSC 56—Advanced Applications Programming **W11, S11, W12, S12, S13 W14, W15, W16, M16, F16**

I start with a discussion of CMPSC 56 both because it is the course I've taught the most often, and the one into which I've put the most significant effort at innovation.
The role of CMPSC 56 in the curriculum is to provide a foundation in Java for students that have already had a year of instruction in C++ (via the CMPSC 16/24/32 sequence.) When the curriculum was revised to be C++ centric rather than Java centric, it was argued that students would be poorly served if there were not at least one required Java course in the lower division. Topics include "generic programming, exception handling, programming language implementation; automatic memory management, application development, management, and maintenance tools; event handling, concurrency and threading, and advanced library use."
My approach to CMPSC 56 is centered on an observation that there is a significant gap between the way that students approach programming assignments in academic classes, and what they will be expected to be able to do as professional software developers either in industry, or when contributing to a significant software project for an academic research lab.

We can identify six dimensions along which this gap occurs:

- **what**: well defined, fixed scope projects vs. open-ended projects with evolving scope
- **when**: short time spans (days, weeks) vs. long time spans (months, years)
- **who**: individuals, pairs, small groups (4-6) vs. larger teams
- **why**: to learn something vs. to address a user need
- **how**: ad-hoc tools, vs. professional tools
- **how-big**: small isolated programs vs. complex integrated systems

This gap has been identified in previous work in the Computing Science Education literature, including [Begel and Simon 2008a], [Begel and Simon 2008b, [Radermacher and Walia, 2013], [Nurkkala and Brandle, 2001] (references at end of statement.)

Therefore, my guiding principle for the course has been to offer a bridge from **coding to learn,** towards **learning to code—**that is, from coding as a means to learn an algorithm, data structure, or problem solving approach—towards learning to code as a professional practice, to build a useful and maintainable piece of software to meet a set of needs.

The central innovation that I've introduced is the use of legacy code projects. From the second offering of the course (S11) forward, each student was required to make contributions to an open source project inherited from a previous instance of the course.
Starting with the S13 instance of the course, these projects were maintained in open source git repositories at https://github.com/UCSB-CS56-Projects with each generation of CS56 students adding additional contributions by fixing bugs, adding new features, or refactoring code. These legacy code projects provide a means to move students more in the direction of authentic software development practice on each of the six dimensions identified.

Organizing the course in this manner has required considerable effort beyond what I and the graduate teaching assistants assigned to the course are able to accomplish, and over the six years that I've taught the course, has created an opportunity for 42 upper division undergraduate students to earn upper-division credit through mentoring students in the course, curating the legacy code projects, and performing peer code reviews.

The process of peer code reviews used in CMPSC 56 was refined through my **participation in an NSF-funded CPATH-2 project,** a multi-institution study titled "Broadening Studio-Based Learning in Computing Education". The workshop held in Coeur D'Alene alone with

teleconferences with the study PIs (from Washington State University, Auburn University and University of Hawaii) provided a supportive framework for exploring innovations to this course. (see my Statement of Professional Activities for more detail on this project.)

In support of all of my curriculum development efforts, but chiefly in support of CMPSC 56, I also undertook a **summer internship at a local tech startup company, AppFolio**, during Summer 2014. As the first "AppFolio Faculty Fellow", I asked to be treated exactly as an incoming new hire with a four-year degree in CS would be treated—to have the same on-boarding experience, and be given the same job responsibilities. The idea was to experience a taste of the "real world" for which I am preparing my CMPSC 56 students.

Based on my experiences, I have already made significant changes. I am putting more emphasis on "user stories" as a way of designing software and expressing requirements. I am emphasizing design patterns more. I have introduced git workflows that more closely align with those used by professional software developers.

Yet, I still see additional opportunities to introduce practices and tools into CMPSC 56 to narrow the gap between academic and professional software practices. I hope to introduce some additional components of the Agile methodology, including Scrum and Kanban, as well as more emphasis on testing, test coverage, static analysis, and continuous integration.

My work in progress also includes evaluation and publishing in connection with CMPSC 56. During the 2016-2017 school year, I had two quarters of study leave that coincided with a visit from Michelle Craig, a "teaching-stream" Associate Professor from the University of Toronto. Prof. Craig and I are collaborating on a study (still in progress) addressing the question of whether the efforts in CMPSC 56 are effective at achieving their goals—i.e. bridging the gap between academic programming and authentic software development practices. (My study leave report and statement of professional activity have more detail on this effort).

CMPSC 56 course evaluation data[2], and sample student comments:

		W11	S11	W12	S12	S13	W14	W15	W16	M16	F16	norm*
A	**median**	1	1	1	1	1	1	1	1	1	1	1
	mean	1.3	1.1	1.7	1.2	1.1	1.1	1.3	1.7	1.2	1.1	1.7
B	**median**	2	1	1	1	1	1	1	1.5	1	1	2
	mean	1.7	1.1	1.7	1.4	1.3	1.1	1.5	1.8	1.4	1.1	1.7

* five year course-weighted norm for CoE CS dept faculty ending F16

"I love the way he uses legacy code in this course. Because of it, I have a strong project on my resume, and have gotten interviews because of it." (F16)

"Probably the greatest comp sci course I've ever taken. Most applicable to the real world, most relevant to my interests/reasons for being a CS major, and the most fun and learning I've had in a single quarter" (F16)

"Pleasure to take from grading rubric to material presented, all aspects of this class prove to make CS56 not only challenging but rewarding as well" (F16)

"I feel so much more prepared for internships now because I learned so many useful tools used in industry. This is also the first course where the skills I learned I was able to put on my resume (ant, git, legacy code, design patterns). Studying for exams was tough because there is literally a mountain of material to review. But it is very doable. As for Conrad, I've never seen a professor care so genuinely about his students" (F16)

[2] See "Note on Course Evaluations", p. 2 for a more detailed explanation of questions "A", "B", and the numeric scores.

CMPSC 8—Introduction to Computer Science M09, M10, F10, F13, S14

During the 2008-2009 school year, the Computer Science department undertook a complete revision of the lower division curriculum, replacing six courses (CMPSC 5,10, 20, 30, 50, 60) with seven new ones (CMPSC 8,16, 24, 32, 48, 56, 64), leaving only CMPSC 40 (the course in discrete math for CS) essentially unchanged.
As part of this overhaul, I taught an experimental section of the introductory course for students with no programming experience in a section for non-majors (CMPSC 5NM) using Python instead of Java. Subsequently, the CMPSC 5 course was changed from Java to Python and renumbered as CMPSC 8 for both majors and non-majors. I taught the first offering of this course in Summer Session 2009, the **first Python-based intro course taught at UCSB**.

A significant change in practice for UCSB that went along with this revision was the structure of discussion sections for lower division programming courses. Prior to the revision, these had been organized around supplemental lectures delivered by graduate TAs. Subsequent to this revision, each of the lower division programming courses had **discussion sections based on hands-on closed labs with programming assignments, with TAs support.** Accordingly, a major part of this transition was the development of a larger number of small programming assignments designed to completed (or at least started) in weekly closed lab sessions.

CMPSC 8 course evaluation data, and sample student comments:

		M09	M10	F10	F13	S14	norm*
A	median	1	1	1	1	1	2
	mean	1.4	1.1	1.1	1.1	1.1	1.9
B	median	2	1	1	1	1	2
	mean	1.8	1.5	1.4	1.2	1.2	1.9

* five year course-weighted norm for CoE CS dept faculty ending S14

"Professor Conrad is an amazing teacher. I knew nothing about programming before I took CS8 but now I am very interested in taking more CS classes." (F13)

"I nominated him for the Academic Senate teaching award. Wonderful professor, wonderful class." (F13)

"Great professor. I had thought about changing my major because of him to CS." (S14)

"I like how when you ask for help, you aren't given just the answer, but rather you are given hints to get to the answer yourself, which makes it stick in your mind more." (S14)

"He was also the only professor who handled the tragedy in May [May 23, 2014] with kindness and sympathy towards the students" (S14).

"After taking this one excellent introductory CS class, Phillip Conrad prepared me, and personally assisted me, in beginning my own 2D video game project." (S14)

CMPSC 16—Problem Solving with Computers I F09, W10, S10, F14, W15

Immediately following teaching the first offering of CMPSC 8, during academic year 2009-2010, I taught the **first three offerings at UCSB of the new course for incoming CS/CE majors, CMPSC 16**. As with CMPSC 8, it was necessary to develop a large number of programming assignments for the labs that accompanied the course. I taught the course three times that year, and many of the materials that I developed that year were subsequently used and adapted by other instructors.

Three years later (late 2013), I worked with two colleagues, Diana Franklin and Chandra Krintz to **examine the CMPSC 16/24/32 sequence and identify ways in which the sequence could be improved**. Feedback from students and instructors was that there was too much material in CMPSC 24, and that moving some material earlier into CMPSC 16, and some material later into CMPSC 32 would provide a smoother path for students. There was also a desire to be able to use only two textbooks across the three courses, and to ease the transition from 16 to 24 by changing the language of instruction in 16 from C to C++.

An outcome of these discussion was that I worked with Professors Franklin and Krintz on a **rolling revision of the CMPSC 16/24/32 sequence**. I taught a revised version of CMPSC 16 in F14/W15, followed by Prof. Franklin teaching a revised version of CMPSC 24 in W15/S15, followed by me teaching a revised version of CMPSC 32 in S15/F16 (discussed further below).

The revised version of CMPSC 16 that I offered in F14 and W15 involved (a) **changing the language of instruction from C to C++** (b) **significantly increasing the pace of the course** with the goal of ensuring that the entire C-subset of C++ (everything up to singly linked using structs, but not included classes) was thoroughly covered before students reached CMPSC 24.

As with the materials developed for the C version, other instructors have made use of the materials developed during the F14 and W15 offering of the course.

CMPSC 16 course evaluation data, and sample student comments:

		F09	W10	S10	F14	W15	norm*
A	median	1	1	1	1	1	2
	mean	1.1	1.0	1.2	1.2	1.1	1.8
B	median	1	1	1	1	1	2
	mean	1.4	1.1	1.4	1.6	1.4	1.9

* five year course-weighted norm for CoE CS dept faculty ending W15

"Prof. Conrad is the best professor I've had at UCSB. I took his CS8 class in Spring 2014 and it was the decisive factor in me deciding in me to attempt to switch to Computer Science as my major." (F14)

"Honestly, the best professor I have had so far in my three quarters at UCSB. Unparalleled enthusiasm and talent for teaching, while also challenging and accommodating his students." (F14)
"Professor Conrad is an infectiously enthusiastic instructor. Prior to this class, I had some interest in Computer Science. I found it interesting, but didn't love it. Now, it is something I know I will want to continue doing/using for the rest of my life." (W15)

"This has been one of the most intellectually stimulating and enjoyable courses of this quarter." (W15)

CMPSC 32—Object Oriented Design and Implementation S15, F15

CMPSC 32 is a second year programming course taught in C++ that covers a few data structures/algorithms topics deferred from the pre-requisite course CMPSC 24, the advanced object-oriented programming features of C++ (inheritance, polymorphism), tools, debugging, and some operating system principles (e.g. processes fork/exec, user/kernel space).

There was also a need to increase the size and scope of projects. In F15, I **added a significant project** in which students had to write code to analyze text from the social networking site reddit.com, calculating a statistic known as tf-idf to "signature words" of a particular communities known as subreddits. To make the assignment more engaging, we focused on calculating signature words for the subreddits associated with various campuses of the UC system. This assignment also involved learning how to link with and rely on external libraries (for retrieval of web pages via http, and parsing of JSON data).

The revision I undertook in S15 / F15 included: (1) adding sorting (insertion, selection, heapsort, mergesort, quicksort), searching (linear vs. binary), and hash tables (2) adopting a textbook that can be used in both CMPSC 24 and CMPSC 32 (3) adding additional programming exercises, including ones involving gdb and valgrind (4) moving many programming exercises to the "submit.cs" autograder system to enable students to get more rapid feedback on their progress, and for better scaling of scarce TA resources as our enrollments grow.

Course evaluation data, and sample student comments:

		S15	F15	norm*
A	**median**	1	1	2
	mean	1.1	1.1	1.8
B	**median**	1	1	2
	mean	1.1	1.3	1.9

* five year course-weighted norm for CoE CS dept faculty ending F15

"I have never had a professor that has been as effective in teaching as Conrad. His enthusiasm, demonstrations, homework and programming assignments are all excellent. During lecture he would patiently answer any questions and works always to try his best in being clear as possible in both concrete and abstract matters." (F15)

"Homework keeps me on track, labs are reasonably challenging, and tests cover only what is covered in class... makes himself available in class, during office hours, and online (Piazza is great.)" (F15)

"I have gained a lot of really useful skills that are applicable in my other areas of CS interests... Conrad definitely seemed very invested in all students". (F15)

"He also gives great book recommendations for further inspection and learning" (S15).

"Github is one of the most important things I learned learned during lab. I am sure it will be very useful (S15)."

"I personally find lower div classes in CS to be slow and repetitive, and consequentially [sic] lose interest, but Conrad does other things to make the class and his teaching useful to me." (S15)

CMPSC 40—Foundations of Computer Science S08, F11, F12, W13

CMPSC 40 is a course in Discrete Math for Computer Science. The most important learning outcome is for students to become comfortable with reading and constructing formal proofs. It also covers propositional and predicate logic, set theory, functions and relations, counting, mathematical induction and recursion.

My most important innovation in CMPSC 40 was an adjustment to the discussion section model. The usual model is to have students sit through yet another lecture, albeit one led by a TA rather than an instructor. I replaced this with a series of hands on exercises that are designed to be done in pairs or small groups. These would typically involve some warm up exercises to ensure that students were familiar with notations and definitions, leading up to practice with writing proofs.

I also tried, throughout CMPSC 40, to help students see the connections between the theory and the practice—to draw lines from the concepts we were covering in class to problems in scheduling, routing, and programming language implementation.

Course evaluation data and sample student comments:

		S08	F11	F12	W13	norm*
A	median	1	1	1	1	2
	mean	1.0	1.1	1.1	1.0	1.9
B	median	1	1	1	1	2
	mean	1.3	1.5	1.4	1.1	2.0

* five year course-weighted norm for CoE CS dept faculty ending W13

"I never missed one of Professor Conrad's lectures because he re-instated my passion for computer science." (S08)

"Professor has been very good at turning a theoretical subject into practical application". (S08)

CMPSCCS 1A—Computer Programming F09, F10, F11, F12, F13, F14
CMPTGCS 1A—Computer Programming F15, F16

This course is the Fall introductory course for incoming CCS Computer Science Majors. It changed designators in 2015 when the degree was renamed to Computing for administrative reasons.

The incoming course for CCS Computer Science/Computing majors is one of the most challenging experiences I've ever had as an instructor. A few things suggest the course should be easy to teach: the students are well motivated, hand selected through a rigorous vetting process, and the course is small: 8-12 students.

Two primary difficulties arise. The first is the vast disparity in the students' prior experience with computing. Some students have been programming, and in some cases, engaging in self-study in topics such as type theory and machine learning—for many years. Other students have discovered a passion for CS relatively recently and have great potential, but much less education and experience. The second is the ambitious learning goal that is set for the course, namely to cover all 30 weeks of material from the CoE curriculum (all of CMPSC 16/24/32) in 10 weeks, and have students ready to take upper division and/or join a research lab by the start of Winter Quarter.

I have found the following strategies to be effective: (1) Differentiated instruction—recognize that to a large extent, the "course" will consist of 8-12 independent studies that are being undertaken by a community of learners. (2) Focus on teaching learning strategies more than content. Since it is relatively rare to find a topic that is appropriate to the learning stage of the entire class, instead, focus on strategies for learning the content, and leave the learning to the students themselves. For many undergraduate learners, this would be ineffective, but CCS students are chosen through a detailed admission review on the basis of this precise characteristic. (3) Create as many opportunities for 1-on-1 private interaction as possible.

I continue to work towards finding ways to create common shared learning experiences that students at many levels of preparation can find rewarding, while providing enough individualized learning opportunities to ensure each student has an appropriate level of challenge—and crucially, so that as soon as possible, each student reaches or exceeds the level of preparation necessary to enter the College of Engineering CS curriculum at an appropriate entry point.

CMPSCCS/CMPTGCS 1A course evaluation data, and sample student comments:

		F09	F10	F11	F12	F13	F14	F15	F16	norm*
A	median	1	1	1	1	1	1	1	1	1
	mean	1.0	1.5	1.1	1.3	1.2	1.0	1.0	1.0	1.3
B	median	1	1	2	1	1	1	2	1	1
	mean	1.0	1.5	1.6	1.3	1.2	1.4	1.6	1.0	1.3

* five year course-weighted norm for CCS faculty ending F17

"Phill is great at teaching by example, and linking what we're working on to the real world. The labs and [the project] are very well explained online through their step-by-step format" (F11)

CMPSCCS/CMPTGCS 1A sample student comments (continued)

"Professor Conrad has a great presence in the classroom. He commands attention while keeping the atmosphere open for discussion and friendly to questions." (F12)

"Professor Conrad is very knowledgeable about computer science and knows how to explain things in effective and exciting ways. I feel like I understanding things like memory management that before baffled me. I enjoyed his enthusiasm and choice of topics and I hope to have other classes with him later. Sitting in on his class when I was in high school was one of the reasons I decided to go here [UCSB] and join CCS." (F12)

"Near the beginning of the quarter, I realized that Computer Science was not the discipline I wanted to devote my life to. Despite this, Professor Conrad was extremely respectful and helpful in every part of the process, understanding that the field is not for everyone. He helped me through Comp Sci, which grew increasingly difficult for me as the quarter progressed. In addition he helped me with my attempts to change majors so that i could pursue my true passions. Professor Conrad is more than just a teacher; he's a mentor, a friend, and oh-so-much more." (F13)

"CMPSCCS 1A has been a very well organized course, and topics discussed are very clearly conveyed by the instructor. The work expected from us is very clear, which at the same time being open-ended. The open-endedness highly encourages thoughtful work, and I feel I put more effort into my assignments because of this fact. On occasion, course sessions go beyond just 'introduction to programming', and focus on working together, and philosophical topics in Computer Science regarding the future of the field" (F14)

"I thought this class was extremely well taught, given the wide variety of students' knowledge coming into this class. He obviously had to teach the basics of C++ for people like myself, but he also kept it at a level for more advanced people so that they weren't slacking... I don't think there was a single day that he wasn't excited to teach." (F14)

"Phill can take complex subject matter and explain it in ways that are understandable and interesting, often drawing analogies to more familiar (and humorous) situations. Despite making class entertaining and light, he still places importance on the subject matter and stays organized and on schedule for class that covers a lot of material. I feel this class has left me with the best possible preparation for my future courses." (F14)

"Phill did a spectacular job teaching the basics of C++ to my class, which contained students of extremely variant levels of knowledge in computer programming... He also did a splendid job of providing real-world applications for the subject matter covered in class." (F14)

"The enthusiasm Conrad teaches with made the course and the material very memorable. His examples, whiteboard charts & graphs, and explanations were great, and helped me really understand things." (F15)

"This course was a joy to be in. I loved the spontaneousness of the classes, and the amount of interesting and complex topics mixed into the beginner material. Professor Conrad knows his stuff and truly cares about our education and development." (F15)

"The instructor shared his knowledge of the subject matter very effectively, and gave us more nuanced information that we would have gotten out of a large lecture by pursuing questions.

The presentation of the course material was made very interesting and engaging.

The professor did a good job of encouraging thoughtful original work and in fact through one of his lectures gave me an idea for my lab project next quarter.

The instructor was also extremely helpful with all matters relating to explaining various elements of college student expectations, the College of Creative Studies, and course planning." (F15)

CMPSCCS 1L—Programming Lab F09, F10, F11, F12, F13

Prior to Fall 2009, CMPSCCS 1A and CMPSCCS 1L, though nominally separate classes, were largely taught with no differentiation between them—the same group of students enrolled in both and one flowed immediately into the other. (This is still the case when 1B and 1L are taught in Winter quarter by a different instructor.)

When I began teaching 1A / 1L in Fall 2009, I separated the roles of the classes completely—with 1A serving as more of a traditional programming course (covering the material from CMPSC 16/24/32) and 1L serving as a projects course.

My goal for this part of the curriculum was to promote and support the "creator" culture among CCS Computing Students.

My requirements for CMPSCCS 1L were simple—although I did not have this vocabulary at the time, the format was quite similar to what is done at an Agile "standup" meeting:

- Every participant must have a well-defined programming project—one with a set of specific, explicitly identified user needs that the project will meet when complete.
- Every participant must, each week: (1) Report on the progress made since the previous week, which must be non-zero, even if it is only a single line of code.
  (2) Report on any places where the student is stuck (3) Set a goal for the following week.

During the time I offered the course in this style, the course attracted many more than just the CCS incoming students—returning CCS students and CoE students were also regular participants. Since F14, I've turned this course back over to my colleague, Murat Karaorman, who has, to some extent, continued the course in this style, i.e. as a project course that complements the more structured curriculum of CMPSCCS/CMPTGCS 1A.

CMPSCCS/CMPTGCS 1L course evaluation data, and sample student comments:

		F09	F10	F11	F12	F13	norm*
A	median	1	1	1	1	1	1
	mean	1.1	1.2	1.0	1.1	1.1	1.3
B	median	1	1	1	1	1	1
	mean	1.3	1.2	1.1	1.2	1.1	1.4

* five year course-weighted norm for CCS faculty ending F13

"I felt that Phill was incredibly knowledgable. For any project that a student came up with, Phill had something meaningful and informative to contribute... The class was very flexible--presentations were student-driven. Phill made sure that we stayed mainly on topic and knew when to move on to the next discussion point... He has an immense capacity for thinking up new projects and pushing students to go further." (F11)

"This course is my favorite course that I could ever imagine. I'm learning so much because I have the freedom to pick a topic I'm passionate about, and because I have several other computer science majors in this class whose knowledge I can draw on. This class is my ideal learning environment." (F12)

"Provides the perfect environment for collaborative CS projects. It's fun and enlightening." (F13)

"Excellent course. Really pushed the boundaries of my CS skills (even though I was a little averse to it at first.) There was a lot involved in the project but he was very helpful in explaining what needed to be done and also further directions to take the project in." (F13)

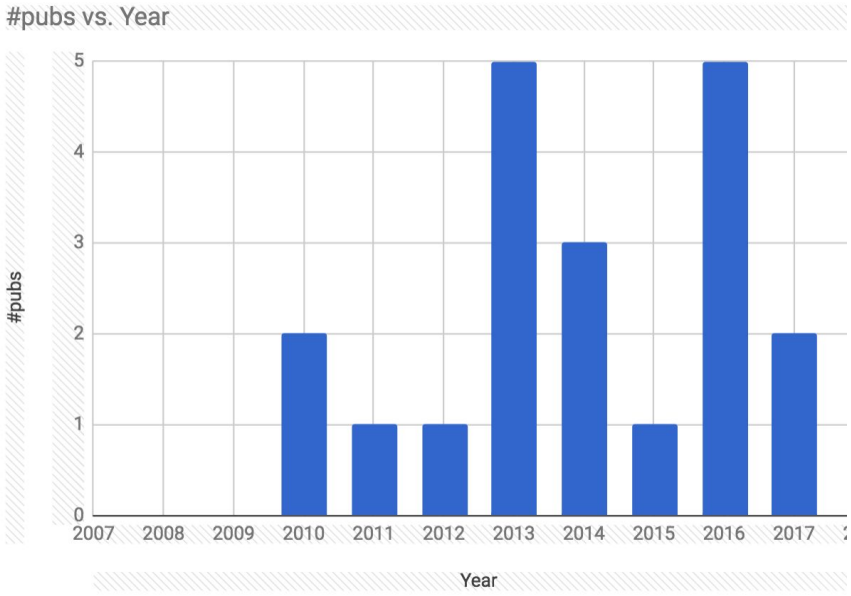
CMPSCCS 130H—Research Methods in Computer Science W09, W10, W11
CMPSCCS 140—Research Methods in Computer Science W12, W13, W14
CMPSCCS 10—Faculty Research Seminar W12, W13, W14, W15
CMPTGCS 10—Faculty Research Seminar W16

This course has gone through a variety of forms and names over eight years that I've offered it, but the key purpose has remained the same throughout: **to promote undergraduate participation in research in Computer Science.**

More specifically, the key outcome being pursued in this course is help CS undergrads from both CoE and CCS to find a research lab that will allow them to join, and make a significant enough contribution to eventually be a co-author on a significant publication.

While both CoE and CCS students have participated in this seminar, I currently only have data related to CCS CS/Computing student co-authorship on publications.

Indeed, since 2009 when this course was first offered, the number of CCS undergraduate co-authors on peer-reviewed publications, has steadily risen, as shown in the chart below.



This list of papers with CCS undergraduate co-authors since 2007 (enumerated below) includes conferences that according to http://www.guide2research.com/topconf/, are among the top conferences in CS: ICWSM, ASPLOS, KDD, IEEE SSP, NDSS, VLDB, and USENIX SEC among others.

The primary unifying theme of CCS's programs is that its students are expected to be creators, not merely consumers, of new scientific knowledge and artistic/creative work. In the STEM disciplines, the expectation is that this takes the form of collaborations with faculty on research projects, leading, in many cases, to co-authorship on peer-reviewed publications. When I arrived at CCS nine years ago, this was an established norm in other STEM disciplines in CCS (Math, Biology, Chemistry and Physics), student participation in research labs was not yet established as a norm in the CCS Computer Science program (as it was known at that time).

To address this need, I introduced a new course, the form of which evolved over the period during which I taught it. The course takes advantage of an existing resource, a Faculty Research Seminar course offered as a graduate level seminar each winter quarter. The graduate version is a weekly 1 hour seminar, in which two faculty members each deliver a 30 minute talk about their research—essentially a pitch to the first year Ph.D. students as to why they might want to work in their research lab.

During the first six years that I taught the course, I constructed it as a four unit upper division course. Attending the weekly faculty research seminar along with the graduate students was one component. To that, in order to round out the course into a four unit upper division course, I added:

- a discussion component, where we would invite the faculty that had presented talks to stay for an additional 30 minute discussion with only the undergrads,
- a lecture / project component, in which, to earn a full four units, students were required to choose a research topic, perform a literature search, prepare an annotated bibliography, and present a research talk from an existing paper. I offered lectures explaining the process of research in Computer Science by example, and guiding students through each of these projects.

Due to the variable unit structure of CCS, students were able to opt-in, opt-out of the various components of the course. Over time, it became clear that participation in the seminar and discussion was highly valued by the vast majority of CCS students, as well as quite a few CoE students who also sought out the course as a way to make connections with faculty for research collaborations. Accordingly, starting with W12, I offered a separate option to register for a 1-2 unit lower division course that included only the weekly seminar and discussion, rather than having students register for the 4 unit version, and then earn less than full units.

Starting with W15, I retired the upper division version completely due to low enrollment and completion rates—while the students completing the course gave it high marks, very few students were choosing to undertake the significant workload. Instead, I retained the version of the course that, on the basis of student enrollment, participation and results, was bearing the most fruit. That course was continued by Prof. Omer Egecioglu during my W17 study leave, and I plan to continue to offer it (or arrange for it to be offered by a colleague) as long as it continues to be valued by students and faculty.

Publications with CCS CS/Computing Student Co-Authors Since 2009

CCS CS/Computing student co-authors are listed in bold. Work by students in the five year BS/MS program is included when the student joined the research lab while still in undergrad status.

Mai Elsherief, Elizabeth Belding and **Dana Nguyen**. #NotOkay: Understanding Gender-based Violence in Social Media International AAAI Conference on Web and Social Media (ICWSM) 2017, May 2017, Montreal, Canada.

Joseph McMahan, Michael Christensen, Lawton Nichols, **Jared Roesch**, **Sung-Yee Guo**, Ben Hardekopf, and Timothy Sherwood. 2017. An Architecture Supporting Formal and Compositional Binary Analysis. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17). ACM, New York, NY, USA, 177-191. DOI: https://doi.org/10.1145/3037697.3037733

C. Krintz, R. Wolski, N. Golubovic, **B. Lampel**, V. Kulkarni, B. Sethuramasamyraja, B. Roberts, and B. Liu. SmartFarm: Improving Agriculture Sustainability Using Modern Information Technology. (PDF), KDD 2016 Workshop on Data Science for Food, Energy, and Water (DSFEW), August, 2016

Daniel Spokoyny, **Jeremy Irvin** and Fermin Moscoso del Prado. Explicit Causal Connections between the Acquisition of Linguistic Tiers: Evidence from Dynamical Systems Modeling. Proceedings of the 7th Workshop on Cognitive Aspects of Computational Language Learning. August 2016, Berlin.

SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis. Y. Shoshitaishvili, R. Wang, C. Salls, N. D. Stephens, M. Polino, **A. Dutcher**, J. Grosen, S. Feng, C. Hauser, C. Kruegel, G. Vigna. Proceedings of the IEEE Symposium on Security and Privacy (SSP 2016). PDF.

Driller: Augmenting Fuzzing Through Selective Symbolic ExecutionN. D. Stephens, J. Grosen, C. Salls, **A. Dutcher**, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, G. Vigna. Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS 2016). PDF

Wei Dai. Randomness Extractors - An Exposition. Rose-Hulman Undergraduate Math Journal (pdf). Vol. 17, Issue 1, 2016.

Kyle Dewey, **Jared Roesch**, and Ben Hardekopf. 2015. Fuzzing the Rust Typechecker Using CLP (T). In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (ASE '15). IEEE Computer Society, Washington, DC, USA, 482-493. DOI=http://dx.doi.org/10.1109/ASE.2015.65

Kyle Dewey, **Jared Roesch**, and Ben Hardekopf. 2014. Language fuzzing using constraint logic programming. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering (ASE '14). ACM, New York, NY, USA, 725-730. DOI= http://dx.doi.org/10.1145/2642937.2642963

Ben Hardekopf, Ben Wiedermann, **Berkeley Churchill**, and Vineeth Kashyap. 2014. Widening for Control-Flow. In Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 8318 (VMCAI 2014), Kenneth Mcmillan and Xavier Rival (Eds.), Vol. 8318. Springer-Verlag New York, Inc., New York, NY, USA, 472-491. DOI: http://dx.doi.org/10.1007/978-3-642-54013-4_26

Vineeth Kashyap, Kyle Dewey, **Ethan A. Kuefner**, **John Wagner**, **Kevin Gibbons**, John Sarracino, Ben Wiedermann, and Ben Hardekopf. 2014. JSAI: a static analysis platform for JavaScript. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014). ACM, New York, NY, USA, 121-132. DOI:http://dx.doi.org/10.1145/2635868.2635904

Xiaohan Zhao, **Adelbert Chang**, Atish Das Sarma, Haitao Zheng, and Ben Y. Zhao. 2013. On the embeddability of random walk distances. Proc. VLDB Endow. 6, 14 (September 2013), 1690-1701. DOI=http://dx.doi.org/10.14778/2556549.2556554

Gang Wang, **Tristan Konolige**, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. 2013. You are how you click: clickstream analysis for Sybil detection. In Proceedings of the 22nd USENIX conference on Security (SEC'13). USENIX Association, Berkeley, CA, USA, 241-256. Metadata at ACM Digital Library, PDF

Madhukar N. Kedlaya, **Jared Roesch**, Behnam Robatmili, Mehrdad Reshadi, and Ben Hardekopf. 2013. Improved type specialization for dynamic scripting languages. In Proceedings of the 9th symposium on Dynamic languages (DLS '13). ACM, New York, NY, USA, 37-48. DOI= http://dx.doi.org/10.1145/2508168.2508177

Vineeth Kashyap, John Sarracino, **John Wagner**, Ben Wiedermann, and Ben Hardekopf. 2013. Type refinement for static analysis of JavaScript. In Proceedings of the 9th symposium on Dynamic languages (DLS '13). ACM, New York, NY, USA, 17-26. DOI= http://dx.doi.org/10.1145/2508168.2508175

Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, **Paulo Almeida-Tanaka**, Brynn Kiefer, Chelsea Laird, Felicia Lopez, Christine Pham, Jessica Suarez, and Robert Waite. 2013. Assessment of computer science learning in a scratch-based outreach program. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 371-376. DOI:http://dx.doi.org/10.1145/2445196.2445304

B. R. Goldsmith, E. D. Sanderson, **D. Bean**, and B. Peters. Isolated catalyst sites on amorphous supports: a systematic algorithm for understanding heterogeneities in structure and reactivity. *J. Chem. Phys.* 138, 204105 (2013).

Adam Lugowski, David Alber, Aydm Buluç, John R. Gilbert, Steve Reinhardt, Yun Teng, **Andrew Waranis**. A Flexible Open-Source Toolbox for Scalable Complex Graph Analysis. Proceedings of the 2012 SIAM International Conference on Data Mining. DOI= http://dx.doi.org/10.1137/1.9781611972825.80

Berkeley R. Churchill and Edmund A. Lamagna. 2011. Summing symbols in mutual recurrences. In Proceedings of the 17th annual international conference on Computing and combinatorics (COCOON'11), Bin Fu and Ding-Zhu Du (Eds.). Springer-Verlag, Berlin, Heidelberg, 531-542. DOI= http://dx.doi.org/10.1007/978-3-642-22685-4_46

Kyle Klein and Subhash Suri. Multiagent Pursuit Evasion, or Playing Kabaddi. International Workshop on Algorithmic Foundations of Robotics (WAFR), Singapore, Dec 13-15, 2010.

Kyle Klein and Subhash Suri. Robot Kabaddi. 22nd Canadian Conference on Computational Geometry (CCCG), Winnipeg, August 9-11, 2010.

CMPSCCS 130H/140 course evaluation data, and sample student comments:

		W09	W10	W11	W12	W13	W14	norm*
A	median	1	1	1	1	1	1	1
	mean	1.2	1.3	1.0	1.0	1.0	1.0	1.3
B	median	1	1.5	1	1	1	1	1
	mean	1.0	1.5	1.0	1.5	1.0	1.3	1.4

* five year course-weighted norm for CCS faculty ending W14

"Phill helped students get connected to professors for potential lab positions". (W09)

"Prof. Conrad and the course have helped to setup up a working relation [sic.] with Professor Sherwood in Undergrad research. The course has provided a large breadth of knowledge in CS I wouldn't have otherwise been aware of. Being able to work on presentation skills in CS has helped me greatly." (W09)

"Great class! Phil [sic.] provided insight into how research is performed at a research university such as UCSB. This has been invaluable in my dealings with faculty." (W09)

"Conrad was able to engage the students in every lecture and always provide thoughtful insight. In addition, he did a great job motivating and encouraging us to start our own original work in computer science research" (W11)

"The course is a fantastic addition to the curriculum. It permits students to see practical applications of the skills they are learning, as well as letting students meet professors and learn their research, allowing easier opportunities to participate in research." (W13)

"On Mondays [when Prof. Conrad lectures] we do new and valuable exercises and we look deep into the academic field of CS research. Fridays are also great, as the [CoE CS] professors brought in to talk provide great insight and inspire students to dig deeper into their respective fields". (W14)

CMPSCCS/CMPTGCS 10 course evaluation data, and sample student comments:

		W12	W13	W14	W15	W16	norm*
A	median	1	1	1	1	1	1
	mean	1.1	1.1	1.1	1.4	1.2	1.3
B	median	1	1	1	1	1	1
	mean	1.1	1.0	1.3	1.2	1.0	1.4

* five year course-weighted norm for CCS faculty ending W16

"Hearing about the different professors research was enlightening and opened my eyes to new possibilities in computer science. I had not previously understood the depth and breadth of the field" (W13)

"I have a much better understanding of what goes on in the CS department. One of the lectures even led me in part to apply for a summer internship". (W14)

"The weekly meetings cover a wide range of topics, and being able to ask questions of the researchers is very beneficial, since it is a rare opportunity. I've found where I want to do research, and I imagine other students are the same." (W15)

"It gave me an open door to talk to some of the professors I would want to work with" (W16)

Other CCS Courses

Brief summaries of the other CCS courses I've offered, in chronological order.

CMPSCCS 130G—Web Applications Development S08

This course was an overview of web application development was based on a similar course, I had taught three times previously at the University of Delaware, and was my first experience with teaching in CCS, primarily based on Java Servlets.

CMPSCCS 130G—CCS Admissions Webapp Competition W09, S09
CMPSCCS 140—CCS Website Redesign W09, S09

This pair of 2-unit courses, each of which spanned two quarters in order to make a full "four unit course", was offered over W09 and S09. Both courses offered an intrinsic motivation, namely, an opportunity to contribute to a web site or web application that might one day represent and serve the college.

The **CCS Admissions Webapp Competition** course was an opportunity for students to design and implement a candidate replacement system for the CCS Online Application for Admission, using any web development technology of their choosing. Twelve CS majors enrolled, evenly divided between CoE and CCS students—eleven of whom stayed with the course for both quarters.

The **CCS Website Redesign** course was targeted at a broad CCS student audience of primarily non-majors. The primary focus of the course was what Richard Saul Wurman (designer, and co-founder of the TED talks) calls "information architecture", i.e. organizing the information on a website with a structure that makes it easy for users to find what they need, and applying that idea to a reorganization of the CCS Website, which was long overdue for an overhaul. It attracted students from CCS Literature, Math, and Music Composition.

CMPSCCS 130G—Software Development for Education S10

This was a projects course run in a manner similar to the model of the CMPSCCS 1L course discussed earlier—with the difference that rather than students choosing their own projects, they were focused on three specific projects related to education.

CMPTGCS 20—Computing Education Methods for K-8 W11
CMPSCCS 140—Projects in Computing Education S11
CMPSCCS 140—Developing K-12 Curriculum with Scratch S12

All three of these course were related to the Animal Tlatoque project, discussed in more detail on the accompanying Statement of Professional Activities. The W11 and S11 courses were ones in which undergraduate students worked one-on-one in an exploratory manner to teach Computing through the programming environment Scratch (the same program used in Animal Tlatoque), and make notes about what was effective and what was not effective. By contrast, the S12 version focused on building a set of organized materials around specific learning objectives—materials which were later used during the 2012 Animal Tlatoque camp.

INT CS 120—Sound, Image and Computation S13

This course, co-taught with CCS Music Composition faculty member Dr. Leslie Hogan, offered students an opportunity to explore the relationship between sound and visual imagery, while learning concepts of music, music composition, computational thinking, and computer programming.

Dr. Hogan provided instruction in Music Composition, while I provided instruction in the programming language Scratch. I also arranged for guest lectures by Charlie Roberts, a Ph.D. candidate from Media Arts and Technology, who presented information on Gibber, his

framework for in-browser live coding and audio performance based on JavaScript (and part of his dissertation work).

To prepare for the course, both Dr. Hogan and I participated in an NSF-sponsored two day workshop titled Computational Thinking through Computing and Music, led by faculty Gena Greher and Jesse Heines at U Mass Lowell (June 21-22, 2012). The workshop provided materials that Dr. Hogan and I used as a starting point for developing our own curriculum.

CMPSCCS 130E—Serious JavaScript S14

This course was an exploration of JavaScript as a serious programming platform. As noted in the course description: JavaScript is a programming language that used to get no respect, but by 2014, it had APIs for threading, 3-d graphics, digital sound synthesis and geolocation, just to name a few. We covered TDD with QUnit, web workers (multiple threads), JSDoc for JavaScript documentation, WebGL for graphics, and the Web Audio API, as well as frameworks such as JQuery, underscore, and node.js.

CMPSCCS 20—Introduction to Computer Science for Non-Majors S15
CMPTGCS 20—Introduction to Computing for Non-Majors S16

This course provided an opportunity for CCS CS students to take an intro CS course. Many find it difficult, as all UCSB non-CS majors do, to get enrolled in CMPSC 8, due to high demand for the course, and limited availability of slots for students for whom the course is not a requirement.

For this course, I adapted material from CMPSC 8 to the CCS model of instruction. The most important adaptation was to allow students to complete the programming labs at their own pace, with the possibility of earning variables units in the course based on the number completed.

The course was well received, with an enrollment of 23 students in S15 and 20 students in S16. Between the S15 and S16 offering, all seven majors in CCS apart from Computer Science/Computing were represented.

CMPTGCS 140—Agile SAAS Development W16

This course offered students the opportunity to participate in one of two software development teams using the Agile software development methodology (see http://agilemanifesto.org/) to contribute to open source software projects—specifically Project Awesome and/or Anacapa Grader. I brought in Dillon Kearns, a UCSB CCS alum, who currently works as an Agile consultant for Santa Barbara area tech companies including Invoca and Procore to run our Agile retrospectives, and offer advice on how to organize and work effectively. The 11 students—5 from CCS Computing and 6 from CoE CS—made contributions to both projects, and gained valuable experience with Agile processes including standups, Scrum, writing user stories, story estimation (points poker), backlog grooming, pair programming, testing, and agile retrospectives. Devops issues such as deployment on Heroku, version control through github, and continuous integration via Travis CI were also covered.

CMPTGCS 130G—Digital Audio Programming Techniques S16

The goal is of this course was to help participants develop an understanding of the basic principles upon which digital audio programming is based, including but not limited to: time and frequency domain representation of signals, sampling, filters, additive and subtractive synthesis, amplitude and frequency modulation.

The course attracted 14 participants, including nine CCS CS/Computing majors, one CoE CS major, two CCS Math majors, and one student each from CCS Math and Physics.