Teaching Statement—Phillip T. Conrad
Prepared for review for promotion to LSOE—08/11/2011

Teaching Philosophy

My teaching philosophy can be expressed in a simple statement: the most important factor in what a student learns is what the student does, not what the teacher does.

Therefore, my approach to teaching is to give primary attention to the activities the students can engage in that will best help them achieve the course goals and outcomes, rather than what I as the teacher will do. My focus is on choosing what reading, what homework assignments, and what projects will present them with opportunities to learn. Lectures play an important, but secondary role—that of preparing students with knowledge, skills, and motivation to be able to complete those activities. It may be a cliche, but it is true—the most important learning takes place outside the classroom.

I believe that the first step towards effective teaching is to recognize that we cannot directly cause learning to take place—rather, we can only create favorable conditions for learning. To put it another way, I see my main teaching goal as doing everything in my power to make it more likely that students will choose to spend more time on, pay more attention to, and give more energy to the learning activities I've offered to them, versus the many other ways in which they may choose to spend their time.

In this battle for the attention of our students, we have two types of tools we can employ: extrinsic and intrinsic motivators—and I use both. The extrinsic motivators include grades and/or units, or showing how learning the material may lead to productive employment or financial gain. Intrinsic motivators include encouraging passion for the material, and showing how the material can be intellectually satisfying, or even, in some cases—fun!

My appointment—and my teaching load—is divided 50/50 between two very different contexts: (1) lower division courses in the College of Engineering where class sizes typically are between 60-100, and where the course objectives are fixed as part of an ABET accredited curriculum (2) a variety of lower and upper division seminar-style courses in the College of Creative Studies, where course objectives are often more flexible, and may vary according to the individual abilities, interests, and goals of the enrolled students. Therefore, the next two sections address each of the contexts separately. I conclude with a section describing the integration of teaching and research in both contexts.

Facilitating Learning in College of Engineering Courses

Two limiting factors on whether a student learns in a course are: (1) does the student read the textbook (and/or other assigned readings), and (2) does the student attend the course meetings (lectures and discussion sections). Both of these provide the necessary foundation for students to be able to complete the assignments and projects where most of their significant learning will take place.

In my College of Engineering lower division courses I try to promote both behaviors by (1) ensuring that every reading assignment is given simultaneously with a specific homework assignment tied to that reading assignment, (2) requiring that all homework assignments be submitted in person during class meetings, (3) assigning homework that must be turned in at nearly every class meeting.

It is not enough, however, to simply achieve attendance in lectures, if students are not engaged—that is, paying attention and participating. Therefore, I do my best to make lectures as interactive and engaging as possible. Here, I employ several techniques, including (1) Learning students' names—even in large classes such as the 109-student CS16 section. Students seem to be more willing to participate when I've made it clear to them that I "know"
them. (2) Asking for a student from the "third row" or the "fifth row" (for example) to answer a question. This avoids putting individual students on the spot—the pressure is diffused among a group of five or more students—but it does keep everyone more "on their toes". (3) Asking for volunteers to solve problems on the computer in lecture. I frequently do live coding in class with my laptop hooked up to the projector. I've noticed, however, that if rather than coding myself, I have a student volunteer do the coding, there seems to be a higher degree of interest on the part of the other students. So, I now employ this technique frequently.

Engaging Assignments

To increase the amount attention, time and energy put into project work, I strive to design assignments that engage students' creativity. One example is the use of individualized graphics assignments, where each student must write two functions (in Python, C, or Java, for example) that "draw" a particular simple item—this could be a pencil, a desk, a football, a basketball court, the logo of a sports team. The students enjoy picking out something that matches their interests. The crucial rule, however, is that what each student draws must be different from what every other student in the course is drawing. (I use an online forum in UCSB's course management system (Gauchospace) where students register their drawings to enforce this.) This assignment seems to tap into a reservoir of creative energy that is not present with more conventional programming assignments. What is particularly rewarding also is that students often take on challenges far more difficult than I would have assigned to them, and end up—voluntarily—exploring aspects of computer science that go beyond stated course objectives.

I have now used this approach for assignments in three different languages: Python (CS5NM, CS8), Java (CS10, CS56), and C (CS16). C presents a special challenge: while Python and Java have built in graphics capabilities that are easy enough for novice programmers to grasp, most existing graphics libraries for C require advanced programming concepts that are not appropriate for a course such as CS16. To address this, building on an idea developed by Tobias Höllerer for his CS60 class, I collaborated with undergraduate students from both CoE and CCS Computer Science (through the framework of my CS1L class from Fall 2009) to develop a library of simple 2D graphics primitives that requires no knowledge of C that is beyond the scope of CS16. The project covers the concepts of developing functions, arrays, structs, and file I/O—and especially the concepts of generalization and function decomposition, while providing a very satisfying and rewarding creative experience. Some of the student work produced can be seen at the website for the course: (http://www.cs.ucsb.edu/~pconrad/cs16/drawings).

This type of assignment also helps to address the issue of improper collaboration, and/or outright plagiarism (direct copying of code), since each student's code must be different (since each is drawing a different object.) When subsequent assignments incorporate these individualized drawing routines, it becomes impossible to simply "copy" code from another student. This does not necessarily completely eliminate improper collaboration, but it does raise the difficulty level, and may be a useful way of nudging students towards better choices.

Pair Programming

Together with my LSOE colleague Diana Franklin, I helped to facilitate the introduction of pair-programming in lower division courses at UCSB, starting with CS10 in Spring 2009, and CS8 in Summer 2009. Since then, pair-programming has become a standard practice in the lower division curriculum at UCSB, and I have used it in every programming course I taught during the 09-10, and 10-11 academic years. There is considerable research about the use of pair programming in introductory CS courses that suggests it is useful for improving learning.
College of Creative Studies Teaching Statement

Specific contributions during this review period

In Fall 2009 and 2010 I had the opportunity to take on the CCS CS1A/CS1L courses—which, along with CCS CS2, a course in Discrete Math as applied to Computer Science, constitute the first quarter of the CCS CS lower division curriculum.

An important consideration for both the College of Engineering CS Department and the CCS Computer Science program in creating the faculty position I now occupy was this: to help create a better working relationship between the two programs. And, one of the key issues in that relationship was a perception that some CCS CS students were arriving in upper-division courses in CS with inadequate foundations and preparation.

The challenge for me in taking on CS1A/1L was to address this concern about fundamentals, while still preserving the *sine qua non* of a CCS education—namely giving students opportunities for creative exploration of their chosen field, limited only by their imagination and willingness to work. My approach has been to take the two courses CS1A and CS1L, which had previously been tightly coupled, and de-couple them into separate courses with separate foci.

The new “prime directive” of CS1A is to ensure that students either have mastered the essential content of CS8, 16, 24, 32, 48, and 56† that they need in order to succeed in upper division courses such as CS130A—or failing that, they are aware of the specific skills and knowledge they still need to acquire before they are ready for upper-division work, and are directed to complete that preparation in CS1B, through independent study, or by enrolling in CoE lower division courses. Thus, the course proceeds at a fast pace through the essential content of all of those courses, with the intention of finding the students “growing edges”, and challenging them just at the point where they can learn something new.

CS1L, by contrast is a course where I offer as much freedom as possible, within a "learning community" that supports one another. The only requirement is to have a software development project that one is working on, to make incremental weekly progress, and to provide both a written and oral report each week on progress made, and goals for the coming week.

My General Philosophy of CCS CS teaching

The key philosophical difference between a CCS education and a more traditional undergraduate education is there is an emphasis on what students *create*—that is, students should not be passive recipients of art and science that already exists, but rather engaged in the act of *creating* new art and science.

My understanding is that, although *creativity* is valued—and necessary for creating new art and science—the emphasis in CCS on the *act* of creating something new, rather than a philosophy of creativity. In the context of Computer Science, then, creative activity in CCS can mean the act of creating software—and at a certain level in a student’s development, this is an appropriate interpretation of creative activity. Once we fully embrace the role of a science discipline in CCS, we should point our students more in the direction of research in *Computer Science*—that is, contributing to the body of knowledge about Computing in some way—a direction I’ve tried to emphasize in all my activities in CCS Computer Science.

This emphasis on creative activity fits in very well with my philosophy of teaching that it is *what the student does* that most affects learning outcomes.

---

† CS40 material is covered in CCS CS2, and CS64 material is covered in CCS CS1B.
In the context of CCS, I see my role as a teacher as one of guiding students towards creative activities that will enhance their learning—first in the area of software development, to enhance their programming skills, but ultimately towards participation in undergraduate research in collaboration with faculty.

**Integration of Teaching and Research**

One of my teaching goals is help CS undergraduates in both CCS and CoE take full advantage of the opportunities afforded them by the fact that they are pursuing their undergraduate degree at a first-rate research university.

While this is an important goal for both CoE and CCS, it is especially critical for CCS, since the key philosophical difference between a CCS education and a more traditional undergraduate education is there is an emphasis on what students *create*. One of the reasons that CCS calls itself a “graduate school for undergrads” is that CCS students are expected not only to study and master art and science that already exists, but also engage in the act of *creating* new art and science.

To this end, I designed a new upper division course (CMPSCCS 130H) called “Research Methods in Computer Science” that gives students an opportunity to hear lectures by ladder faculty about their research, as well as learning about the process of doing research in Computer Science. Each of the times I’ve offered this course (W09, W10, W11) both CCS and CoE CS students have participated, and each time it has led to undergraduate students establishing research relationships with CoE faculty.

In addition, I’ve offered an upper division course called “Software Development for Education” (CMPSCCS 130G) that has offered both CCS and CoE students opportunities to develop software and skills that are related to the Animal Tlatoque project (supported by an NSF grant) and the Design/Play/Teach project (for which I have applied for funding, in collaboration with two faculty from UCSB’s Gevirtz Graduate School of Education.) Three of the students from the Software Development for Education course in Spring 2011 did end up being hired as paid staff for the Animal Tlatoque project during Summer 2011, and are currently participating in research that will offer co-authorship opportunities on publications.