STATEMENT OF TEACHING PHILOSOPHY AND INTERESTS

WILLIAM CHRISTIAN BENTON

The Surprising Idea

When I was an undergraduate, my teachers helped me to discern and synthesize the common themes, tools, and puzzles of the liberal arts curriculum. I came to see deep connections between the skills I developed in my computer science courses and the problems I examined in my other major coursework. Understanding programming language semantics facilitated understanding philosophy of natural language, and vice versa. My ability to construct beautiful and correct programs grew in parallel with my ability to employ rhetoric in the service of arguments and essays. As I have grown professionally, my own teaching has been informed most directly by the following surprising idea: *Computer science study hones essential intellectual tools for students in every discipline.*

This idea has borne fruit: I have enjoyed a pleasant and successful tenure as a teaching assistant and lecturer while conducting my dissertation research. The Computer Sciences department at Wisconsin has entrusted me with a broad range of teaching assignments in introductory, senior-level, and graduate courses. My efforts have been recognized with excellent student evaluations, enduring mentoring relationships with former students, and selective departmental and university-wide teaching awards. I have introduced undergraduates to the excitement of computing, nurtured young computer scientists, and advocated the discipline to a wider academic community that is too often confused about what computer science *is* – and thus is surprised that our field has anything to offer artists, biologists, and classicists, among others.

Teaching Interests

I am capable of and interested in teaching a wide variety of courses. I have direct experience lecturing for an introductory programming course as well as a senior-level operating systems course. I have also served as a teaching assistant and grader for a graduate-level theoretical programming languages class. I would be glad to teach survey courses in programming languages or systems, as well as courses related to my research experience, including courses on concurrency, compilers, logic programming, virtual machines and computer architecture. I am also interested in leading seminar courses on topics in these areas or on cross-disciplinary topics related to my other interests, such as computer music or non-classical logics. Finally, I am passionate about collaborating with other faculty to develop new curricula, especially for introductory courses; these are, after all, our first contact with students and a bridge between our discipline and the rest of the academy.

Principles & Goals

My primary goals as an instructor are to:

(1) Model a critical and creative approach to problem solving,
(2) Nurture students as they develop intellectual clarity and rigor,
(3) Build a vocabulary of tactics, principles, and patterns that are applicable not only to programming, but to other intellectual puzzles, and
(4) Create an environment that encourages all students to succeed in and enjoy the challenges of computer science.

The course I have taught most frequently at Wisconsin is *Introduction to Object-Oriented Programming*. This is the first course in the CS major, but it is also popular with non-majors – some because it fulfills program requirements and some because it satisfies a curiosity for the subject matter. In seven terms lecturing for this course, I have taught fine students with a wide range of experiences, backgrounds, and motivations. Just as they come to the classroom with different sets of skills and desires, my students leave the classroom and set out on a variety of different paths. Almost all will *use* computers in some way after the course, but they will do so in different ways: some will be computer scientists; some will be programmers by vocation; some will use computers to solve problems in another discipline; and yet others will simply need to treat problems in everyday life in a thoughtful and systematic way. Given these circumstances, I have developed an approach that makes the material accessible and useful to all my students.

The Challenge

Teaching programming is, at its essence, teaching students a new way to solve problems: instead of merely figuring out how to solve a particular case and moving on to something else, programmers must essentially design and construct a machine that is general enough to solve a family of related problems. As a result, students must think about problems in a deeper and more formal way than they may have previously, they must achieve sufficient clarity to specify instructions precisely, and they must exercise their facilities for abstraction and logic to ensure that the pieces they have designed fit together well and act as advertised. These skills are applicable across disciplines, since they are also necessary for good writing: constructing an elegant and correct program from various cooperating subroutines is akin to constructing a clever and persuasive argument from evidence. Student programmers must also develop some design sense – like architects, their work should be functional, safe, and beautiful. However, unlike buildings or essays, programs and their requirements may change frequently, so the programmer has a concern that only rarely affects the architect or rhetor: namely, she must design a program that will admit extensions and improvements with minimal perturbation.

The challenge of teaching programming is this: while the skills that students will build are almost universally applicable to future studies, daily work, and scholarly endeavors, the tools of the trade – the abstract concepts and concrete grammars of programming languages – may seem obscure or confusing at first. Students must learn the concrete tools in order to absorb the concepts, and learning the skills that will make a student into a good programmer must happen concurrently with learning the myriad details of the syntax, semantics, and rules of a particular programming language. As a result, students may be overwhelmed by the details and miss the joy of building their problem-solving abilities along with correct and elegant programs.

Engaging Multiple Learning Styles

I HAVE MET this challenge in the classroom in several ways. My lectures, course materials, and interactions with individual students are designed to make abstract concepts understandable and relevant, and to make students explicitly consider how to apply the skills they have learned in service of their work away from computing. I use several tactics to introduce new material: orally, with clear and direct prose; visually, with handouts and animation; and conceptually, with analogies to the tangible and familiar. Through these teaching styles, I am able to reach students with a variety of learning styles. However, I have also found that each tactic introduces students to a new facet of the material:

(1) Introducing material with straightforward prose models the way I expect my students to engage the material: to understand concepts well enough to define them with precision and clarity. In class discussions of especially intricate material, I often ask students to find the most succinct accurate explanation for a term; these conversations can be even more valuable than their conclusions.
(2) Presenting programming concepts visually eliminates confusion and reinforces subtle details. Such presentations can also serve as models: for instance, my students learn to picture exactly how particular statements change the state of the computer. This skill is a crucial part of the programmer's craft, and the act of picturing program state strengthens a student's grasp on the relevant concepts.
(3) Finally, explaining concepts with analogies gives students a framework for understanding concepts, encourages abstraction and cross-disciplinary thinking, and gives the instructor an opportunity to employ humor – making students laugh is a sure aid to attention and retention.

Practicing Together

I ENCOURAGE STUDENTS to apply new skills and ideas outside the domain of computer programming. For example, as I introduce abstraction and object-oriented modeling I often ask my students to identify the essential properties of a familiar concept – *What makes this thing a "chair?"* Another successful exercise asked students to consider the design of real-world things: How do we interact with them? Which things have a clearly demarcated set of responsibilities, and which do not? (One of the best student-supplied examples of the latter was a refrigerator that included a web browser.)

By making explicit connections to the familiar and encouraging discussion and participation, I am able to create learning activities that involve all students – even those who believe that they "aren't good at this stuff." These sorts of activities build community and encourage students to explore and engage the material more deeply. I have found that creating communities in this way generates enthusiasm for not only the subject matter but also for the kind of problem-solving that computer scientists prize, and it works well both for students who enter the course enthusiastic about computing and for students who might otherwise fall to the margins.

Future Challenges

I BELIEVE THAT teaching and mentoring students is one of the most exciting and challenging responsibilities of the profession. Creating an environment that nurtures passionate computer scientists demands a dynamic and deliberate approach. I look forward to creating an effective learning environment and am excited to train the next generation of computer scientists, programmers, and critical thinkers from across the academy.