Andrew Begel: Teaching Statement

I love to teach. For this simple reason, teaching has been and always will be a significant part of my life. I strive to help students understand the subject matter, encourage them to think deeply about the core concepts of the class, and most importantly, excite them to learn. A successful teacher excites students through a combination of good teaching skills, a passion for learning, and a desire to share that passion with others. Over the course of my undergraduate and graduate teaching and research career, I have discovered that I possess these qualities.

My teaching skills come from many hours of teaching experience and from pedagogy classes I have taken. In addition to discovering my strengths as a teacher, I have learned how to develop new curricula, write engaging assignments, design fair tests, and evaluate and reflect on a course and its execution in order to improve it in the future. My innate passion for learning has led me to take every opportunity to expand my knowledge about science and engineering. My desire to share this passion with others has led me to teaching, where I can motivate a new generation of scientists and engineers to be as eager to learn as I am. It has also led me to create new pedagogical environments to help students learn about science and programming. Combining my two loves, teaching and epistemological research, I involve undergraduates in my research projects so that they might experience first-hand how fascinating computer science can be outside of the classroom.

Teaching

I have taught several classes as a teaching assistant (TA), guest lectured for introductory classes, and taught programming to high school students in an after-school program. Not only have my experiences nurtured my enthusiasm for teaching, but they have also taught me many valuable practical classroom skills, including the importance of preparation and practice, setting high, but realistic expectations, and learning how to improvise when the lesson plan goes awry. I devote a portion of every class session I teach to "Fun With *X*", where I can show students the silly, fun, and fascinating aspects of the material that they are learning. One technique I have used to great success in the "Fun With *X*" section is the kinesthetic learning activity (KLA), a physically engaging classroom-based activity that teaches computer science concepts. For example, in the introductory programming in Scheme class that I taught, we played Cons Cell Jeopardy, where a group of students would come to the front of the class to act out a box-and-pointer diagram, Twister-style. The other students would work in groups to try to come up with the Scheme code that produced that data structure. The game was engaging for all of the students in the class and conveyed some ideas about recursive data structures that may otherwise have been too abstract for some of them. Along with my colleagues, lecturers Daniel D. Garcia and Steven A. Wolfman, I conducted a special session on KLAs at the annual SIGCSE Computer Science Education conference in 2004. We presented several fully completed, debugged activities that teachers could use in their own classes and then facilitated teachers working in groups to design their own KLAs around a variety of computer science subjects. The session was enthusiastically reviewed; thus we presented it again in a longer workshop form at the 2005 SIGCSE conference.

Course Development

I have developed two courses from scratch, one on programming language design and implementation, and the other for first-time teaching assistants to make them the best teaching assistants they can be. Through my enrollment in the department's teaching minor, I took part in a class that taught us how to create a computer science class. We learned how to develop the syllabus, homework assignments, tests, and projects, choose grading schemes, manage teaching assistants, develop new course extensions and evaluate the course. I designed a programming language course that takes emphasis away from the front end of the compiler to make more time available for language design, intermediate representations, control flow and data flow analysis, optimizations, code generation and runtime systems.

In designing the course, I tried to imbue it with three design ideals, inspired by an introductory computer science course I took as an undergraduate at MIT. On a basic level, my course was defined to teach how compilers and programming languages are implemented. Since it was an introductory course, however, it had a second purpose – to introduce students to different kinds of programming language paradigms and their varied compilation, interpretation and implementation strategies. But for me, its most salient objective was to teach students about the "zen" of programming languages – guidelines for successful language design and implementation that apply not only to the most trivial toy language, but to large-scale programming systems as well. I tried to design this course with the structure of good literature; each carefully chosen reading, assignment and test question should have many nuances in order to enable students of all abilities to obtain something unique from their learning experience. Students who are ready to learn the higher-level messages will do so, and those who are not will simply learn to build a compiler and runtime system.

With my colleague, lecturer Daniel D. Garcia, I developed and taught a completely new version of our teaching techniques course for new teaching assistants. Each week, we would concentrate on the pragmatics of teaching, with frequent detours into educational philosophy. We devoted a portion of each class to "group therapy," providing a forum for TAs to talk about their week and get advice on how to deal with immediate, sticky problems. To encourage reflection, we asked TAs to write each week in an online journal about topics like how they prepare for section, what it takes to be a great teacher, and how to understand their students' needs. A big part of learning to teach involves getting feedback on how you are doing. So, with the help of the Berkeley GSI Teaching and Resource Center, I developed our department's first TA video reflection program. We videotaped each TA's section and asked him or her to watch it and write down his or her impressions. Afterwards, I watched the tape with the TA, and together we discussed his or her teaching strengths and weaknesses, and as well as concrete ways he or she could improve his or her teaching. This exercise often led into a discussion of teaching philosophy and made the TA more aware of how the students were reacting to different actions taken during class.

Invited by Berkeley's GSI Teaching and Resource Center, I have led four workshops at the twice-yearly Orientation and Teaching Conference to teach EECS graduate students enough about teaching to handle their first week of TAing. I have also been invited four times to speak to the students in our department's teaching techniques course about what makes a great TA.

Educational Research

Thinking about teaching and learning influence not only my teaching career, but also my research. For my dissertation, I developed a programming environment that increases the accessibility of programming to people with disabilities and makes programming easier by improving the computer-supplied feedback that programmers use as they construct their program. We have deployed a program editor based on this environment in almost every semester's offering of the compiler course here at Berkeley. Many of the editor's features were designed to help students learn to program in Cool, the language used in the course. In their feedback, students rated our environment 5.5 out of 6, appreciating the syntax highlighting and indentation features the most. Parse and type checking errors were also helpful to students learning Cool syntax and semantics. I believe that my dissertation work could be extended to help novices learn to program. The simpler, more natural syntax afforded by voice-based programming can shorten the length of time that students spend learning syntax, enabling them to concentrate more on language semantics.

While in college and throughout graduate school, I collaborated with Mitchel Resnick at the Lifelong Kindergarten Group at MIT's Media Lab and Eric Klopfer in the Teacher Education Program at MIT to develop the StarLogo programming environment. StarLogo is a parallel version of Turtle Logo designed to help high school and college students learn about modeling complex systems, such as ant colonies foraging for food, or traffic flow on a highway. First, students develop an algorithm to describe a system's behavior. Then, they program a StarLogo model to implement their algorithm. Finally, they test and validate their model through experimentation. I have helped conduct summer StarLogo workshops for

secondary school students and teachers in collaboration with the Santa Fe Institute. We paid careful attention to making abstract systems concepts, such as dynamic equilibrium and local and global information flow, more concrete for students through participatory simulations. Very similar to kinesthetic learning activities, students each play a role in a complex system, for example, by behaving like a bird in a flock; together the students' combined behaviors cause the emergent properties of the system to become apparent.

Currently, with Eric Klopfer, I am developing a new version of StarLogo aimed at increasing student interest in programming through development of games. New StarLogo features such as visual programming and 3D turtle graphics will help enable students to build high-fidelity modern-looking video games. Early field testing with high school students indicates that our goals are being realized. Students report that the 3D world offers a more realistic visualization of turtle interactions and simulation context. They also like the puzzle piece design of the graphical programming language because it helps eliminate syntax errors and enables them to more easily learn the vocabulary.

Undergraduate Research

It is important to me that undergraduate students be involved in computer science research. Undergraduate research can make the theory that students learn in their computer science classes more concrete, fascinating, and relevant than any problem set, test or project. When I was a sophomore in college, I began working on a research project with Mitchel Resnick through MIT's Undergraduate Research Opportunities Program (UROP). Most UROP experiences are meant to last a semester or two; mine has lasted 11 years. My role in the project has evolved from a developer to a designer, and more recently to a research collaborator. Being involved in UROP was, without a doubt, the most significant experience of my undergraduate career, and as a graduate student, I made sure to offer research opportunities to others. With Eric Klopfer, I have advised 35 MIT undergraduates on the StarLogo project. In my own dissertation work at Berkeley, I have mentored 16 undergraduate researchers, seven of whom have gone on to graduate school. I will enthusiastically continue this tradition as a faculty member, and will encourage my graduate students to do the same.

Teaching Interests

I believe my enthusiasm for teaching, as well as my extensive experience in course development and TA training have prepared me well to design and teach my own courses as a faculty member in computer science. I am excited to teach any introductory computer course for majors and non-majors such as programming, algorithms and data structures, and computer architecture. My graduate research experience qualifies me to teach advanced courses in programming languages, compilers, program optimization, parallel programming, human-computer interaction, and software engineering. I would also like to teach a capstone project course and a course on social issues and computer ethics. On the graduate level, I want to teach courses on programming language design and implementation, advanced parsing and optimization, assistive technologies, and software development techniques. I also want to teach a course on teaching techniques for teaching assistants (and faculty members) and a course to help aspiring student teachers learn how to design and implement their own computer science courses.