Eunsuk Kang Teaching Statement

I am excited about teaching undergraduate courses that touch on any aspect of software development, including software architecture, requirements, programming languages, compilers, and software reliability methods (such as testing and verification). At the graduate level, I am eager to develop and teach research-oriented courses in software design methodologies, formal methods, and an interdisciplinary seminar to explore potential synergies between software engineering and control theory.

Teaching Software Engineering Software development, in practice, is often a messy, non-linear process, with vague and frequently changing customer requirements, and a multitude of design options and trade-offs to consider. I believe that it is important to not only teach students the most fundamental concepts in computer science, but also help them develop knowledge and skills necessary to manage a complex engineering project. I am especially interested in exploring curriculums that balance conventional projects (containing well-defined problems and solutions) with more open-ended ones where students are encouraged to elaborate on an initial problem definition and explore alternative solutions.

As a teaching assistant for introductory and advanced software engineering courses at MIT, I had an opportunity to teach and contribute to this type of curriculum. In particular, for *6.005: Elements of Software Construction* (a mandatory course for CS majors), I was responsible for creating a major course project called *ABC Music Player*. In this project, the students were asked to construct a Java program that plays music files in a format called *ABC*. Beyond this basic requirement, they were given complete freedom with respect to design choices, such as methods for parsing musical notes, handling errors in input files, and dealing with underspecification in the official ABC documentation. The students were evaluated on the quality of design reflections, including the analysis of trade-offs between design alternatives that they considered, the rationale behind their final decision, and a discussion of challenges encountered during the project. Based on course evaluations and personal conversations, the students found the open-ended nature of the project initially challenging, but ultimately enjoyed and appreciated the experience of working with design freedom. Since its inception in 2008, the project remained part of the latest offering of 6.005 in Spring 2016.

Research Mentoring I consider classrooms as a great place to get undergraduate students excited about academic research. During the second year of my undergraduate studies, I took an introductory course in logic and its applications to computer science, where I fell in love with the simplicity and power behind the idea of formally reasoning about programs. To further explore this interest, I participated in several undergraduate research projects in formal methods, eventually continuing on as a graduate student in the same area. Given my own experience as an undergraduate researcher, I plan to incorporate discussions of the cutting-edge and well-established research into lectures where possible, and encourage students to engage in research outside of the classroom.

I personally enjoy working with students on research projects: During my PhD, I have mentored 5 Master's students and 3 undergraduate researchers. I am also working closely with two PhD students, at the University of Michigan and the University of California, Berkeley.

Teaching Formal Methods One of the major obstacles to a widespread adoption of formal methods is a high learning curve often associated with verification tools. Given my experience working with and teaching a modeling language called Alloy (including a tutorial given to developers at a Linux workshop[1]), I am interested in exploring ways to convey the value of formal methods without overwhelming beginners with technical details. In particular, I plan to develop a curriculum that emphasizes (1) potential benefits of formal methods without a lot of manual, upfront investment (e.g., automatically generating test cases instead of fully verifying a system), and (2) domain-specific tools with an input language that is approachable to those with a basic background in programming.

[1] https://lwn.net/Articles/522970/