Teaching Statement

Leonidas Lampropoulos
University of Maryland, University of Pennsylvania

My teaching philosophy is to keep students *engaged* and *motivated* during lectures, foster *collaboration* between them as much as possible, and *adapt* my approach to their individual skills and needs.

Teaching Experience. I am currently teaching **CMSC 631 - Program Analysis and Understanding** at UMD, a graduate level course with 25 registered students, with diverse backgrounds from senior undergrads to early year PhDs and from both Math and CS backgrounds. This course covers theoretical and practical ideas for modeling and analyzing programming languages with a focus on mechanical reasoning in the Coq proof assistant. Out of this course, all students should learn the basics of programming language metatheory and, more importantly, they should improve their programming skills by being able to write *precise specifications* about any piece of code. This is evaluated throughout the course, through a sequence of hands-on verification assignments, starting from simple facts about natural number arithmetic and scaling up to proving functional correctness for imperative programs using Hoare logic. It will also be evaluated through a collaborative final project, where students are asked to develop, test, and verify a medium sized software artifact of their choice.

I based this course on the popular Software Foundation series of mechanized textbooks. It is the first time anywhere that such a course will also cover material from my recently released fourth volume in the series, **"QuickChick: Property-Based Testing in Coq"**. This volume brings together elements of testing and verification under the same roof, teaching the benefits of specification-driven development. The material in this fourth volume was based on a summer school module with the same title that was offered in the **DeepSpec Summer School in 2017 and 2018**, with a large diverse audience of early-year graduate students.

Before that my experience with teaching was small but extremely rewarding, presenting material in seminar-level courses and leading lab recitations. One particular experience that stands out is my teaching assistantship during the second year of my PhD. The relevant course was the introductory computer science course at UPenn (CIS 120), where students are exposed to programming in OCaml and Java. During that course I got to experience a course (re-)design process for the first time: working closely with the professor of the course, Benjamin Pierce, I updated all homework assignments, extending their scope and improving their content, with one exercise being re-invented from scratch. I also co-designed all exams for the course to reflect the newly added homework material. For this effort I received **UPenn's Teaching Practicum Award for 2013**.

Teaching Approach. To keep students *engaged*, I aim to keep my lectures interactive, encouraging in-class discussion and using quick quizzes to stir up conversations about potentially challenging course aspects. For example, while teaching CMSC 631 this semester I make extensive use of the Turning Point app to conduct such quizzes. Many of these quizzes are drawn from the Software Foundations textbook, which the course is based on, but I try to improve and expand upon them

in every opportunity. Some of these improvements have already been accepted in the development branch of Software Foundations and will be used in the next release.

To keep students *motivated*, I try to give real-world applications of the material being covered or draw analogies with current research topics. For instance, in CMSC 631 we cover a very simple imperative language and reason about the correctness of programs written in it. The same techniques and ideas, albeit presented here in a much simpler setting, are actively being used in practice (e.g. in the Verified Software Toolchain for reasoning about C programs).

I also want to encourage *collaboration* between my students. I myself always found that working with peers was more rewarding and productive than working alone, and I try to foster the same teamwork spirit in my students. In CMSC 631, we use the Piazza system as a forum where students are encouraged to both ask and answer questions about course material. Moreover, collaboration is also recommended for the final project of the course, where a small group of people formalize an interesting piece of software.

Finally, I strive to *adapt* my teaching approach to student's individual strengths and weaknesses. I hold regularly scheduled office hours, but also keep an open-door policy where students can come to me for help with both course material they're struggling with and with course topics that they're really interested in and would like to explore further.

Advising Approach. The same idea also extends to my mentoring style. I was fortunate enough to experience different styles during my graduate studies at UPenn and UMD, both from my own advisors and second-hand through my friends' shared experiences. A weekly meeting, with clear small short-term goals and an equally clear long-term vision of the work being conducted, serves as a great "default" approach. However, I have found that different researchers thrive under vastly different circumstances. For example, I personally enjoyed my most productive PhD research days with a more flexible, "on-demand" meeting structure: I met with my advisor, we talked about the greater problem we were addressing, proposed approaches that we could take to solve it, and discussed any issues or results that had arisen in the meantime. I then went off and came back later, within 1-2 weeks, to set up a new meeting and repeat. My best friend required the exact opposite approach to be productive: bi-weekly short "status" meetings with clearly stated day-to-day tasks. I believe part of the job of a mentor is to tune their approach to each particular student, working with them to figure out whether a more flexible or structured approach will keep both happy and productive.