Statement on Teaching

Preetha Chatterjee

My graduate experience at the University of Delaware has influenced and continues to shape my career. I had several opportunities to teach and mentor both early graduate and undergraduate research students, and mentoring students has been a very rewarding part of my graduate education. Here, I describe my teaching experience, teaching philosophy, and research mentoring as I launch my academic career.

Teaching Experience

As **sole instructor**, I taught the second programming course (CISC181) for computer science and computer engineering majors at the University of Delaware during the 2019 10-week summer session. This course covers the principles of computer science illustrated and applied through programming in an object-oriented language (Java), and assumes students have already taken an introductory programming course in either Python or Racket. I created and presented lectures, developed course syllabus, assigned labs, created assignments and exams, and supervised a graduate teaching assistant for this course. My class was small with 12 students, which enabled an increased interaction with students and allowed me to get to know the strengths and weaknesses of my students well, and adapt my teaching accordingly.

Additionally, I have taught graduate courses as a **substitute instructor**, where I facilitated the classes in the absence of the regular instructor. I have conducted lectures and facilitated interactive class sessions for: 'Intro to Computer Science Research (CISC 367)', 'Communication Skills for CS Researchers (CISC 667)', and 'Advanced Architecture/Software Systems: Text Analysis for Software Engineering (CISC 879)'. CISC 367 prepares undergraduates to conduct computer science research by performing literature reviews, identifying important research problems, problem solving/progress tracking, designing/conducting evaluation studies, etc. CISC 667 provides mentoring to graduate students in writing computer science research papers, proposals, and dissertations through focused writing exercises and critiques. The students also learn how to prepare for research posters and presentations, and research interview talks. CISC 879 provides graduate students an opportunity for in-depth exploration of topics in text analysis for software engineering.

I have served as a Lab Instructor and a **Graduate Teaching Assistant (TA)** for many courses. I was a TA and Lab Instructor for introductory computer science courses 'General Computer Science for Engineers (CISC 106)' and 'Introduction to Computer Science II (CISC181)'. CISC 106, which is taken by every non-CS student in the college of engineering, covers the principles of computer science illustrated and applied through programming in Python. CISC 181 is the introductory Java course where I led weekly lab sessions, during which I reviewed the week's lecture material and added my own spin on what they had learned. Besides the general TA duties of grading assignments, holding office hours, answering students' questions over email, I also wrote a lab assignment, and led an exam review session for all three sections. I was also a TA for two courses in web programming; 'Introduction to Computer Science with Web Applications (CISC 103)', and 'Advanced Web Technologies (CISC474)'. CISC 103 covers the principles of computer science through programming in scripting languages such as JavaScript and VBScript; and CISC474 covers the topics of programming and architecture of web servers. Besides the general TA duties, I have also assisted the students in their projects during the class sessions along with the instructors.

Teaching Philosophy

I have enjoyed teaching since I was an undergraduate, and my teaching philosophy is continuously developing and improving. I especially want to facilitate students' learning of CS knowledge and skills and thus make a positive difference in their lives. I work to design my teaching so my students develop strong foundations in theory, and are able to apply the theory to solving important, practical problems and creating useful technology. In my classes, students learn skills, such as problem solving, writing and evaluating programs, and communication skills that they can use and apply after they graduate. I believe in an inclusive learning environment where everyone in the classroom contributes as a student, teacher, and thinker.

Classroom Techniques: I engage the students' curiosity to learn by creating an **interactive classroom environment** where students are comfortable to ask questions and are encouraged to work collaboratively on solutions to real-world problems. I strive to provide plenty of examples and demonstrations to illustrate the given concept. I look to students to participate in and even drive examples. Their questions and comments help to pinpoint their

misunderstandings and help me determine what points I need to discuss further. In programming-focused courses, I develop code with students, which allows opportunities for discussion of design or implementation decisions. I am neither a strict follower of the board style teaching nor of the presentation style; I believe in creating a balance between the two based on the topic. I have found that asking students "Do you have questions?" during the lecture would rarely gather responses, so I follow it up with "On a scale of 1-10, how much do we understand this?". They respond better to the follow-up question, and if they say < 7, I explain the material again.

Feedback: Because students learn from their and others' mistakes, I provide feedback through detailed comments on their assignments and projects. I aim to grade assignments within a week of their due date and provide written **feedback – both to individuals and to the class as a whole** through web forums such as Piazza and lectures to help them critique common problems. Beyond correctness, I give feedback on a program's elegance, efficiency, and readability because those properties are very important after graduation – when peers and managers need to read, debug, and maintain the code.

Communication: I also believe that students should be able to express themselves and analyze their work in written and oral forms. I emphasize well-written code documentation from the beginning of a course and sometimes include a written component in assignments. Students present their projects to me in **individual and in-class demonstrations** so that they learn effective oral communication skills. During demonstrations, we discuss the student's design decisions and possible alternatives and their tradeoffs.

Hands-on Experience: Since long-term course projects give students more opportunity for **creativity and in-depth knowledge** of a particular subject, I assign a project and a couple of assignments that could be modified and extended for the project in my courses. The assigned project is typically based on application of in-depth knowledge of the material and it pushes students to achieve new results along with a boost in their confidence, such as creating their own designed two-player interactive board game (CISC181).

Assessment: To evaluate student mastery of a concept, I use several different metrics: weekly homework assignments, in-class programming exercises, projects, and exams. I assign **two types of homework**: those where students **develop programs from scratch** and those where I give them **a code base to build on**. The former gives students more freedom in design and requires them to review programming fundamentals; the latter allows students to complete larger projects with more functionality, provides additional examples of good programming styles/techniques, and requires students to read and understand another programmer's code. I have also used a combination of individual and group assignments. I use quizzes to gauge student understanding, to encourage students to review notes before class, as a vehicle for in-class review and discussion, and to expose students to the types of questions to expect on exams. Although exams are important tools to analyze the student's learning, I believe having ungraded fun quizzes garner interest and participation of the students.

Resources: I maintain all the resources of a course on web repositories such as Canvas, which provides lecture notes, in-class code examples, assignments, and other course resources. In the future, I would like to implement the idea of **collaborative lecture notes** preparation which would serve the dual purpose of engaging as well as assessing the students. The students would be instructed to create and extensively edit the lecture notes present on a Github repository. The collaborative notes could be of enormous help for students while preparing for exams. Moreover, the edit history on Github would also enable me as an instructor to assess the levels of interest and understanding of different students in the class, and adapt my teaching accordingly.

Potential Courses

I am most excited to teach graduate-level courses related to my doctoral thesis: data analysis and text mining, and software engineering. Data analysis skills are in high-demand on the job market, and data science technology is constantly evolving. These courses would provide students with coveted skills on the job market, and provide me with potential researchers and collaborators. Beyond these courses, I am qualified and comfortable to teach upper-level undergraduate and graduate-level courses on machine learning, database systems, natural language processing, and empirical research methods. These courses would have a strong research component, where students read and discuss research papers, and work collaboratively on state-of-the-art research projects which would grow their interest in choosing Research as a potential career option thereby benefiting the field of Computer Science with budding talents. I am also interested to teach typical introductory computer courses for undergraduate majors and non-majors such as programming, data structures and algorithms.

Undergraduate Research Mentoring

In the past few years, I **mentored seven undergraduate students** on several successful research projects. I collaborated in designing research projects, worked closely with students to guide them through the research process, designing and implementing an approach, evaluating the approach, and presenting their research in paper and oral presentation formats. Benjamin Gause and Hunter Hedinger helped in implementing and evaluating a set of heuristics to extract the natural language text that is associated with each code segment in a research article. This research was accepted as a full research paper in the 14th International Conference on Mining Software Repositories (MSR) [1]. Qilin Ma developed an in-house search-based research tool (with GUI) written in Python for mining developer discussions on Stack Overflow. Our research group has used this tool for data collection in various experiments. Minji Kong worked on qualitative data analysis on developer communications on Stack Overflow. This research was accepted as a journal paper in the Journal of Systems and Software [2]. More recently, Brian Phillips, Humpher Owusu, and Kevin Mason, participated in a data analysis to understand the quality of information in developer chat communications on Slack. This work is currently under submission at a top conference venue in Software Engineering. I have also been involved in **guiding and working in collaboration with junior graduate students**. We have conducted a qualitative study and co-authored a paper, which investigates the kinds of information that is embedded in different software-related documents [3].

Based on my experience, I have found a few simple **strategies for successful undergraduate research mentoring**. First, the projects must be interesting, easy for the student to understand the necessary background, and have concrete, attainable subgoals. The student should be able to see tangible results – such as an implementation or an experimental study – within the allotted time frame. Before the project begins, I create a design document that describes the background, the problem, and an overview of the approach. The document also enumerates the tasks or subgoals necessary to reach the final goal and how the student can get started: background reading, required software to install, etc. The project always entails a variety of tasks – reading, implementation, writing, thinking, experimentation, analysis – so that students never encounter an insurmountable roadblock. Throughout the project, we review what is working and what isn't. We refine our goals and create timelines for when we expect tasks to be completed, and we keep track of the new ideas that we may investigate later if time permits. Within each task, the student has flexibility in the approach to take, but we discuss the tradeoffs of each approach. I also give feedback about the approach at intermediate steps so that the student learns best practices and does not get stuck going in an inappropriate direction.

The most rewarding aspect of mentoring is to make a positive impact and help shape a student's career. An excerpt from one of my undergraduate mentee's thanking email note: *"...With your help I feel I have accomplished a lot and I have learned more than I ever expected. From the beginning of my time working on this project, you have been mentoring me and teaching me about the research process. You have been incredibly patient and supportive throughout this time. Thanks to you I was able to learn about programming, research writing, and a lot about what I want to do. This whole experience has been an incredible opportunity and has never failed to stay interesting. I'm proud to be able to say that I worked on this project with you and the rest of the team."* At least in part because of research experience, some of my mentees are applying or in graduate school now. I plan to continue to prepare undergraduates for careers in research, to encourage them in academic pursuits, and to nurture talented software developers. I plan to mentor at least 2-3 undergraduate students each year, with financial support available through university and/or government programs (e.g., NSF REU Supplements).

References

[1] P. Chatterjee, B. Gause, H. Hedinger, and L. Pollock. Extracting Code Segments and Their Descriptions from Research Articles. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 91–101, May 2017.

[2] P. Chatterjee, M.Kong, and L. Pollock. Finding Help with Programming Errors: An Exploratory Study of Novice Software Engineers' Focus in Stack Overflow Posts. In *Journal of Systems and Software*, JSS '20, 2020.

[3] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft. What Information About Code Snippets is Available in Different Software-related Documents? An Exploratory Study. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 382–386, Feb 2017.