Teaching Statement

Faisal Nawab
http://nawab.me

The fulfillment of being part of the intellectual tradition of mentoring and teaching is a key reason for my pursuit of an academic career. Evolving and growth as a teacher and mentor is a continuous process. I am fortunate to have started this process as a graduate student working as a mentor and teaching assistant and to continue as an Assistant Professor in UC Santa Cruz.

Courses and Teaching

As a faculty member, my background equips me to teach undergraduate and graduate courses on distributed systems, databases and data management, operating systems and networking. I will also be interested in giving advance seminars and lectures on Big Data systems, data management for emerging IoT and edge applications, systems for data science, and the theory of distributed systems and data management. In addition, I would be happy to teach introductory undergraduate courses.

In UC Santa Cruz, I have taught the following undergraduate and graduate courses. I taught the course **CMPS111: Introduction to Operating Systems** two times. This course introduces students to operating systems concepts such as processes, memory management, file systems, virtualization, and I/O. My interest in providing an authentic experience to students led me to adopt a project-driven methodology where students apply the concepts they are learning on a real operating system kernel (the FreeBSD kernel.) The project is divided into four parts, in each part the student would hack the kernel to implement a new design for one of the kernel's components. For example, one of the projects is to implement a new scheduling algorithm which enables students to get a better intuition on how processes and multi-threading work.

I also taught the **CSE130: Principles of Computer Systems Design** course. This course aims to introduce students to the principles that govern the design and complexity of complex software systems. Specifically, in the course, I aim to familiarize the students with concepts to manage software complexity such as abstraction, modularity, and hierarchy. Then, I tie these concepts to technical foundations and designs to enable building scalable systems. These technical foundations include topics in concurrency control, synchronization, layering, naming, client-server networking and performance. I also adopt a project-driven approach to enable the students to get the intuition of these concepts. In the quarter I taught this course, students built a HTTP server and in four milestones they explored concepts about networking, synchronization, multi-threading concurrency, caching, naming, and performance.

Another course I am currently teaching in UC Santa Cruz is **CSE20: Beginning Programming in Python**. This is the first programming course that is taken by computer science majors. Students in this class are typically ones with no prior knowledge of programming (students have the option to take a test and skip the course if they are familiar with programming.) This introduces unique challenges to teaching the course in addition to its critical nature as this course serves as the foundation for the rest of the student's studies. For these reasons, I have adopted an interactive learning approach. The textbook required by students is provided by ZyBooks, which is an online, interactive textbook that enables the students to learn the concepts of the course via interactive modules that enables students to try out and observe how these concepts are reflected in real programs. Additionally, I adopt this in the classroom, where I live code the examples and programs that pertain to the lecture so that students can get the intuition and breadth of the concepts being taught. To make the course more accessible, I designed the programming assignments to enable students to get immediate feedback on whether they performed the required task. For example, the student would enter their code and an automatic grader would produce the score for public and hidden test cases. This enables students to get more confidence on their approach and solutions.

I also taught two graduate courses. The first is **CSE214/CS277: Principles of Database Systems** which I taught two times. This course introduces students to the principles and foundations of database systems. This includes concepts such as concurrency control, transaction processing, database design, and various query processing topics. I structure the course to have two parts. The first part that takes 6 to 7 weeks out of the 10 weeks of the course is about the foundations of database systems and has a clear structure and readings from papers and textbooks. The rest of the course is about emerging topics related to the course. In the two times I taught this class, the emerging topic for the first one is global-scale and geo-replicated systems and for the second one is byzantine and blockchain-based systems. The other graduate course I taught is **CMPS 290S: Advanced Topics in Computer Systems**. This course targets advanced Ph.D. students and is structured to include discussions of recent research papers and a large research project.

Teaching Strategy and Reflections of Changes in my Classroom Teaching

I strive to be an enabler of students' success in, and beyond the classroom. To achieve this, I apply four key principles in teaching. First, I focus on the *fundamentals* of the subject. Computer science is a dynamic field where the technical details are rapidly changing. Understanding the fundamentals is what will enable students to adapt to such a changing field. Second, I ensure that students understand the *context* of what is being taught. A common difficulty faced by students is that it is not clear "why" a certain concept is relevant and how concepts are related to each other. An end-to-end approach to show how the concepts are manifested in familiar systems and technology can greatly help provide the context and appreciation of the taught concepts. Third, the course must enable students to *independently* learn deep and cutting-edge topics. A course, due to time constraints, only touches the surface of a computer science subject. However, it can teach the tools necessary to enable students to independently learn new topics within the subject. This skill is vital for the students' future careers as they are more likely to face technical problems that were not explicitly taught in class. Finally, a teacher needs to keep students *motivated* by making classes and tasks more entertaining, thought provoking, and interactive.

An important feedback channel that I pay special attention to are students evaluations. I have learned a lot from reading student evaluations and it influenced my teaching positively. One of the changes that were triggered by students evaluations is finding ways to reduce students stress. Specifically, I started opening more channels for dialogue and opportunities to interface with students inside and outside the classroom. One example is that I now start each lecture with an open discussion where students can ask about anything related to the course before we start the lecture. Another change is to find more ways to increase students participation. To this end, I benefited from feedback to encourage participation and always taking students answers with encouragement as a step towards a correct answer.

The wonderful feedback and changes that were influenced from students reviews led me to also implement midterm evaluations—which I did when teaching CSE130 in Fall 2019. Students reacted extremely positively to the midterm evaluation as they have pointed out many changes that can improve their experience and they witnessed how I adapted my teaching accordingly. These changes include streaming the lecture and notes taken during the lecture and finding ways to make the assignments more clear. To answer for the second concern, I started an *assignment clarity committee* which would meet with me before an assignment is released to look it over and point out any confusions or unclear parts. I found these changes so wonderful that I have written about them in my blog[1].

Graduate and Undergraduate Mentoring and Research Lab Supervision

In the two years I spent in UC Santa Cruz, I have started mentoring undergraduate and graduate students. My lab— the EdgeLab—now consists of 5 Ph.D. students in their first or second year. EdgeLab is part of a bigger lab organization called the Languages, Systems, and Data Labs that consists of the labs of four UC Santa Cruz faculty. The EdgeLab Ph.D. members have started embarking on their research and have their work already in the pipeline to be published at this early stage in their studies. To facilitate their success, I have ensured to pair them with senior students from other labs to enable them to observe how research work is performed at a closer level.

In addition to Ph.D. students, I mentored 10 M.S. students in the previous two years (acting as their project committee chair.) Most of these students work with me for two or more quarters on a research project. In some cases, these M.S. students are paired with one of EdgeLab's Ph.D. students to work on parts of the Ph.D. student's research project. The typical M.S. student project consists of a literature survey in an area, identifying a problem, creating a design to solve the problem, building a prototype and a benchmark, evaluating the prototype, and writing a report.

In terms of undergraduate research, I have mentored more than 20 undergraduate students. I manage undergraduate research in two ways. The first is the traditional one-on-one mentorship where a student is given a project and paired with a graduate student. Eight undergraduate students were mentored this way. The second way I mentor undergraduate students is via large group meetings, where all undergraduate researchers are working towards a common goal. In the 2018/2019 academic year, I ran a project with more than 10 undergraduate students to build an Edge-Cloud database prototype. Students divided into teams of 2 to 4 students working on various aspects of the database and one of the teams handled the big picture/project management side of the project. In three quarters this team of undergraduate researchers successfully managed to create a working prototype that we are now using in our lab.

Prior to joining UC Santa Cruz, I also mentored undergraduate and M.S. students. One notable example is Colin Biafore. Colin's work with me was presented in SIGMOD 2016 as part of the Undergraduate Research Competition, and later he was the Computer Science department nominee for UCSB's 2016 Undergraduate Research Award.

Mentoring Philosophy. My goal as a mentor is to train students to do independent and original computer science research, a skill that is not acquired in a regular classroom setting. Such a skill is important for students pursuing both academic and industrial positions, in a field where industry is rapidly adopting, and sometimes leading, cutting-edge research. As a mentor,

[1]Faisal Nawab, "Three Things I Did in Class Last Quarter (That Students Liked)": https://nawab.me/blog/?p=614

I tailor the amount of practical and research work to match the student's background and future career plans. My experience as a mentor showed me that finding the right balance between hands-on and hands-off supervision is important to develop the student's independence and character as a researcher. Also, it enables students to pursue their projects in paths that are more suited for their interests and background. Repeatedly, I have been delightedly surprised by students taking their project to novel directions that I did not anticipate; and often, I ended up learning from the students as they became experts in their work.

Education Technology

Technology and computer science are transforming education and learning. I am particularly interested in two emerging technology trends in the intersection of education and technology:

- *Interactive learning:* Interactive educational tools have the potential to enhance the learning experience. Recently, computer science academics have started implementing interactive textbooks, demonstrations, and visualizations to supplement traditional learning material. I started adopting interactive textbooks (e.g., ZyBook) and plan to imitate and extend such experiences to the upper-division courses I teach.
- *Diversity and inclusion:* Having more *diverse teams* for class projects can promote a teamwork culture that is more inclusive and less skewed to specific groups. This is challenging to implement given the demographics of today's STEM classrooms. However, with online resources and collaborative software, it is now feasible to pair students to form diverse teams even if the team members are in different physical locations and institutions. I plan to pursue engaging departments of fields that have better diversity in terms of gender, gender identity/expression, and sexual orientation. There are many opportunities to pair computer science students with students in other fields, especially in capstone projects and introductory courses that can have real world applications in the sciences and arts.