UC Irvine

Teaching Statement

Syed Fatiul Huq

Students learn software engineering best when they build something with other people and see what it does to the people who use it. After a guest lecture I gave on software accessibility, one student wrote that learning about assistive technologies made the challenges faced by disabled users "more concrete and urgent." My experiences as a teaching assistant and occasional guest lecturer at University of California, Irvine, and an instructor at University of Dhaka, Bangladesh have shaped how I teach. I follow three tenets that make learning more engaging: connecting concepts to a product students build, a collaborative environment, and direct feedback from and examples of people using their product.

Learning from building

As a teaching assistant for the software capstone, software system design and software design courses, I supported multi-week team projects building software products, often for external stakeholders. Designed to iterate on an ongoing project with new weekly lessons, these courses showed me that students engagement rose when theoretical lectures were mapped to tangible outcomes. It provides an avenue for them to apply new knowledge, and better appreciate the nuances and significance of their lessons. For instance, when students designed actual systems, they were able to put to use design concepts such as mindmapping, personas, UML and more. At the end of the quarter, one student came up to say that they introduced "GACIOD" matrices — something they learned in class — to their colleagues, which then became a staple for ideating and validating plans before building any system.

Learning as a team

In the project-based courses I was assigned to, teams worked in Agile sprints, and tracked work in team management software. This format aimed to mimic actual software development practices in the industry. It took students out of their comfort zones and taught them important lessons in teamwork, leadership and management. It also brought about challenges in technical aspects that they had not encountered in individual work, such as concurrent prototyping and programming. When teams ran into trouble sharing and merging code, I taught version control and helped them move their project to Git. When teams went quiet or stalled, I reinforced Agile practices that made progress visible and encouraged taking ownership of their work proactively rather than waiting to be assigned work. Based on my experience in building software, I offered ideas for features and evaluated the prototypes, code and interfaces they produced. I also coached teams on communicating with their stakeholders, where nomenclature and expectations often mismatched.

Learning through real examples

This last tenet has been influenced by my experience as a guest lecturer in HCI and software engineering courses teaching about accessibility. Most students had not encountered accessibility as a concept in the curricula, let alone knew how assistive technologies like screen readers and voice control worked. In the lectures, I demonstrated assistive technologies and showed examples of disabled users navigating applications. Such examples resonated significantly with the students, who, in follow up conversations and evaluations, conveyed their appreciation, surprise and new-found commitment for considering and incorporating accessibility. In weekly reflections, one wrote that inclusive design should be "a standard,

not an afterthought." These experiences have strengthened my conviction to incorporate real world examples and build empathy in students as part of my teaching methods.

Other than demonstrations, I have utilized my expertise of accessibility to write assignments and rubrics for the project courses. I introduced guidelines like WCAG, testing tools like WAVE, and manual testing strategies using screen readers and simulators. It made accessibility something students were tasked with and graded on, not only told about.

Mentoring experience

Outside the classroom, I have mentored undergraduate, master's, and PhD students in research, helping them scope questions, design studies, write protocols, conduct user studies, analyze data, and frame their work effectively. During my master's, I taught Java and Android development to a class of 15 working professionals over a six-month term, an experience that taught me to adapt to learners on non-traditional tracks. I have also participated in the **Mentoring Excellence Program** at University of California, Irvine to better prepare me for future mentorship and advising roles.

My mentorship philosophy, which stems from these experiences, rests on three principles: alignment, inclusion, and tailored support. I begin each mentoring relationship by aligning with my mentee's values, expectations, and experiences. I provide an inclusive space for diverse perspectives and beliefs, across gender, race, ethnicity, and ability. And I tailor my support to my mentee's strengths, recognizing that each individual responds to different levels of directness, attention and freedom.

Teaching in the age of AI

One of the biggest points of contention with the advent of LLMs is how we as educators should treat AI in the classroom. Based on my observations as a TA, and my own use of agentic tools, I believe that LLMs can be harnessed appropriately, without replacing student agency and motivation. Unfortunately, students will use coding agents regardless of policy. I would teach them to treat agent output as a draft to verify and reflect on, instead of adopting without scrutiny. Hands-on work during class hours, and reverting to pen-and-paper tests and oral exams, are interventions I have observed to succeed and will experiment with in my own classes. Furthermore, my research experience with LLM harnesses has given me insight into how purpose-built scaffolds can be developed that can enhance the learning process for students. For instance, programming courses can introduce AI pipelines, so that students, who already use industrial coding tools, can use ones built by us, which will be engineered to prioritize the learning process over simply generating results.

Courses

Based on my educational and research background, I can teach existing courses at the Paul G. Allen School of CSE. This includes courses related to software engineering, human computer interaction and programming, such as CSE340 Interaction Programming, CSE403 Software Engineering, CSE440 Introduction to HCI, CSE443 Digital Accessibility, and more. I can also contribute to special topics in software engineering such as requirements engineering, software architecture, software design, software testing and maintenance, etc.