Teaching Statement - Ryo Suzuki

Teaching Experience

Albert Einstein once said *"information is not knowledge. The only source of knowledge is experience. You need experience to gain wisdom."* His words best reflect my pedagogical philosophy—teaching is not providing knowledge, but providing an opportunity to **apply the knowledge to a real-world problem**. My teaching experiences at the University of Colorado Boulder demonstrate and reflect the importance of this perspective. As a teaching assistant for Prof. Shaun Kane's Fundamentals of Human-Computer Interaction class, I led two weekly design studio sessions with forty students each. The goal of the sessions was to help our students familiarize themselves with the methods learned in the class, including sketching techniques for ideation, conducting user interviews, rapid prototyping, and heuristic evaluations of each other's work. Through these sessions, I aimed to develop their understanding by providing *real-world examples of how the methods are used* and encouraging them to fill the gap between theory and practice through *learning by doing*. Based on the Faculty Course Questionnaire (FCQ), this approach worked well—my session was highly rated (4.6/5), and the students strongly agreed the course was helpful (4.2/5) and that the TA was supportive and respectful of students (4.4/5).

I also served as a teaching assistant for Prof. Mark Gross's graduate-level Soft Robotics class. In contrast to the undergraduate HCI course, this course was a project-oriented class—students were supposed to learn the literature of soft robotics and develop their own robots or actuators by the end of the semester. My main role was to support their research process through apprenticeship. Leveraging my knowledge and expertise in making soft robots, I nurtured their ideation processes by providing relevant resources and I advanced their prototyping processes through my technical support of their software and hardware implementation. While the students had diverse skillsets from differing disciplines (CS, ME, architecture, design, etc), all successfully completed their final projects, one of which led to a peer-reviewed publication [1].

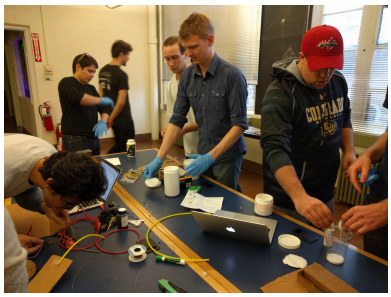

Figure 1: Hands-on activities in the soft robotics class.

Through these differing teaching experiences, I gained a richer sense of how to organize curriculum to best support students.

[1] H. Hedayati et al. Hugbot: A soft robot designed to give human-like hugs. In *Proceedings of IDC*. ACM, 2019

Mentoring Experience

During my Ph.D. and research internships, I have been fortunate to mentor ten talented students at multiple universities. [2] I mentored these students for at least three months with two weekly in-person meetings. I helped them gain experiences in each aspect of HCI research, including the **ideation phase** (e.g., helping them identify an interesting yet solvable research question; collaboratively making a concept video with stop-motion animation), the **prototyping phase** (e.g., guiding them to develop a *minimum viable product* to quickly test their most critical hypothesis), and the **documentation phase** (e.g., revising their writings; demonstrating techniques to make clear figures and videos). The results were very successful. We had six full conference papers, including two CHI and one UIST paper. One of my mentees won the best thesis award at his university based on our two top conference papers, one of which won the best paper award.

Through these mentoring experiences, I learned three guiding principles. 1) **Give concrete and actionable advice**: Abstract, high-level advice may not provide sufficient guidance toward next steps, which often makes students—particularly junior students—get stuck. I always try to give concrete and actionable suggestions to support their forward progress. 2) **Decompose big visions into small tasks**: Students also get stuck with big ideas without having concrete execution plans. In this case, I show them how to break the goal down into several milestones, and further break down each milestone to a task they can complete within a week. 3) **Show instead of tell**: As a mentor, I always try to *demonstrate* how they can change—such as showing them how to revise a paper, showing them how to shoot a video, or showing them how to identify bugs. Students can then learn by comparing their original outputs with my demonstrated examples. As they increase their proficiency, I gradually become more hands-off to encourage them to achieve their goals independently. These lessons have given me confidence in my ability to mentor students and help them grow as active, accomplished researchers.

Novel Tool Development for Programming Courses

Before starting my Ph.D. program, I taught programming (e.g., JavaScript, Python, Ruby on Rails, and Node.js) to a small group of students at the University of Tokyo. Based on this experience, I learned one vital thing: Hands-on support for programming exercises is important, but **it does not scale**. Even with 10-15 students, they continuously encounter bugs and need assistance, which makes it difficult to support all of my students in a timely manner.

[2] A list of students I mentored during my Ph.D. at CU Boulder as well as research internships at Stanford University, UC Berkeley, and the Univerity of Tokyo/Keio University.

- **Kevin Kuwata** (ECE Master at CU Boulder, now Sparkfun X)
- **Zhixian Jin** (ECE Undergrad at CU Boulder)
- **James Bohn** (CS Undergrad at CU Boulder)
- **Chrystalina Pharr** (ME Undergrad at CU Boulder)
- **Ryosuke Nakayama** (Media Design Master at Keio University, now Sony)
- **Takayuki Hirai** (Media Design Master at Keio University)
- **Takumi Murayama** (Media Design Undergrad at Keio University)
- **Michelle Lam** (CS Undergrad at Stanford University)
- **Juan Marroquin** (CS Undergrad at Stanford University, now Microsoft)
- **Adam Ginzberg** (CS Undergrad at Stanford University, now Coda.io)

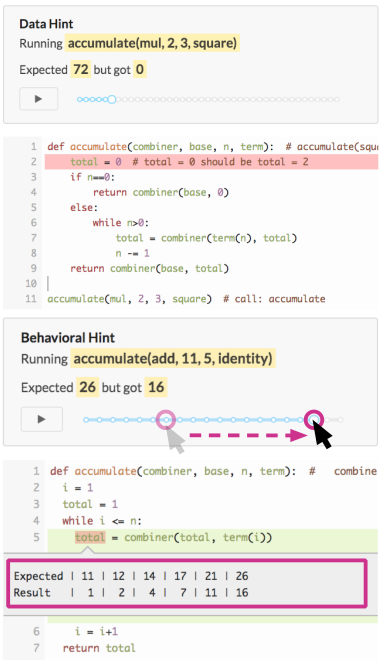

Figure 2: A tool to automatically synthesize personalized hints for massive programming classrooms.

Motivated by this experience, I sought to address this problem. I was fortunate to collaborate with researchers at UC Berkeley to develop and deploy a tool to automatically provide programming hints for introductory programming courses at UC Berkeley, with over 1,500 students. Based on our insights, we published several peer-reviewed conference papers. [3] As a course instructor, I will continue developing our tool to scale personalized support for programming courses.

[3] A list of publications based on our novel tool development for programming courses.

- R. Rolim et al. Learning syntactic program transformations from examples. *In Proceedings of ICSE*. IEEE, 2017;
- A. Head et al. Writing reusable code feedback at scale with mixed-initiative program synthesis. *In Proceedingsof L@S*. ACM, 2017;
- R. Suzuki et al. Tracediff: Debugging unexpected code behavior using trace divergences. *In Proceedings of VL/HCC*. IEEE, 2017;
- R. Suzuki et al. Exploring the design space of automatically synthesized hints for introductory programming assignments. *In Proceedings of CHI EA*. ACM, 2017;
- R. Suzuki. Interactive and collaborative source code annotation. *In Adjunct Proceedings of ICSE*. IEEE,2015

Proposed Courses

Undergraduate Courses:
1. Human-Computer Interaction: Introduces design methodology and interaction design skills to learn the human-centered design approach. The course covers techniques for ideating, sketching, prototyping, and evaluating interactive software and hardware. The course also covers the theory and principles of user interface design.

2. Web Programming: Focuses on the programming and prototyping techniques of web technologies. The course covers a primer on developing web applications using HTML/CSS, jQuery, React.js, WebSocket, and Node.js, then teaches prototyping methods for advanced applications such as mobile apps, AR/VR, and computer vision.

3. Physical Computing: Focuses on lab-based rapid prototyping and the iterative design of functional interactive artifacts. The course covers basic electronics for sensing and actuation, software tools for designing and programming (Arduino, Eagle), as well as prototyping tools for fabrication (laser cutting, 3d printing).

Graduate Courses:
1. Virtual and Augmented Reality: Introduces virtual and augmented reality (VR/AR) with a focus on building spatial computing interfaces. Working in teams, students will ideate, prototype, and test user interfaces. Starting with low-fi prototyping methods, students also learn how to develop interactive applications with Unity or WebVR.

2. HCI Research Survey: Introduces emerging research topics in HCI, with a focus on robotic and tangible interfaces. The topics include tangible interfaces, swarm user interfaces, shape-changing interfaces, 4D printing, inflatable architecture, and haptic interfaces.

3. Human-Robot Interaction: Introduces techniques in human-robot interaction. Students read and discuss primary literature on HRI, including programming by demonstration, human-robot collaboration, and interaction design for communicating with robots.