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 **experience of applying the knowledge to the 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 of forty students each. The goal of the sessions was to help the 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 tried to foster their understandings through *providing real-world examples of how the methods can be used* and encourage 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 the TA was supportive and respects 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 helped their ideation processes by providing relevant resources and helped their prototyping processes through technical support of software and hardware implementation. While the students had diverse skillsets from different backgrounds (CS, ME, architecture, design, etc), all of them successfully completed their final projects, one of which led to a peer-reviewed publication [1].

Through these teaching experiences in different types of classes, I gain a better sense of how to organize the class and how 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 intern, I have been fortunate to mentor more than ten talented students at multiple universities. [2] I mentored these students at least three months with more than two weekly in-person meetings. I helped them gain experiences in every aspect of the HCI research, including **ideation phase** (e.g., help them identify an interesting yet solvable research question, collaboratively make a concept video with stop-motion animation), **prototyping phase** (e.g., guide them to focus on making a *minimum viable product* to quickly test the most important hypothesis), and **documentation phase** (e.g., revise their writings, demonstrate techniques to shoot a video). The results were very successful. We had six full conference papers, including two CHI and one UIST paper. One of the students won the best thesis award at his university based on our two top conference papers, one of which won the best paper award.

Through such diverse and successful mentoring experiences, I learned three guiding principles: 1) **give concrete and actionable advice**: abstract, high-level advice may not provide a guidance of what they should do next, which often makes students, particularly junior students, get stuck. I always try to give a concrete and actionable suggestion so that they can move forward. 2) **break down into small tasks**: students also often get stuck at big ideas without having any concrete execution plans. In this case, I break the goal down to several milestones, and further break down each milestone to the task that they can complete within a week. 3) **show instead of tell**: As a mentor, I always try to *demonstrate* how they should change, such as showing how to revise the paper, showing how to shoot the video, showing how to prototype. They can naturally learn by comparing their original outputs and demonstrated ones. Once they learned, I gradually become more hands-off to encourage them to do by themselves. These lessons gave me a confidence in mentoring students to help them grow as independent researchers.

[2] A list of students I mentored during my Ph.D. at CU Boulder as well as research interns 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)

Novel Tool Development for Programming Courses

Before starting the Ph.D. program, I was also teaching programming (e.g., JavaScript, Python, Ruby on Rails, Node.js) to a small group of students at the University of Tokyo. Based on this experience, I learned one thing: hands-on support for programming exercises is very important, but **it does not scale**—even with 10-15 students, they continuously encounter bugs and need help, which makes me hard to support all of the students in a timely manner.

Motivated by this experience, I have also tried to address this

problem. Particularly, 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, that have over 1,500 students. Based on the insights we gain, we also published several peer-reviewed conference papers. [3] As a course instructor, I also wish to continue developing a tool to scale up the personalized support for programming courses.

[3] A list of publications based on the 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 a human-centered design approach. The course covers techniques for need-finding, 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 programming and prototyping techniques of web technologies. The course covers a primer on how to develop 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: Focus on lab-based rapid prototyping and iterative design of functional interactive artifacts. The course covers basic electronics for sensing and actuation, software tools for design and programming (Arduino, Eagle), and prototype with digital fabrication tools (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 from the low-fi prototyping method, 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 of human-robot interaction. Students read and discuss primary literature on HRI. The topics include programming by demonstration, human-robot collaboration, interaction design for communicating with robots.