CHRISTOPHER J. PARNIN

TEACHING STATEMENT

THE FUTURE OF THE UNIVERSITY

In the coming years, new faculty face a potentially tumultuous and transformative period. The future of the university is being questioned on many fronts. The boldest affront has been on the university’s core role: teaching and educating its students. In particular, the idea of massive open online courses (MOOCs) has lead to speculation that in the extreme case, universities become consolidated into one of a few nationally branded universities. Despite early data suggesting significant flaws with MOOCs, there are certainly many appealing aspects: reduced costs, courses could be made more interchangeable and distributed, and the opportunity to transform educational technology’s infrastructure.

In the area of software engineering, there are many unique challenges centering around both the relative newness of the field and the rapidly evolving nature of software. Software engineering is still a young discipline, with just under 50 years passing since the coining of “software engineering”. Although software underlies almost every facet of today’s world and infrastructure, we have not yet been able to construct a discipline that creates software in a manner is consistently repeatable or teachable. For example, we still have little evidence as to why we teach students to use agile software processes or object-oriented programming languages. As programming becomes a more pervasive skill, having strong empirical evidence and cognitive theories is essential for a better understanding of how to teach it.

In the following sections, I first describe my passion for teaching and how my 2.5 years of teaching experience and 10 years of industrial experience have shaped my teaching philosophy. Next, I describe how I am well-positioned for tackling the unique challenges education will face based on my empirical approaches to computing education, and exploration of education technology. Finally, I detail my plans for teaching and mentoring students in my new role.

MY TEACHING PHILOSOPHY

Teaching is incredibly rewarding and often the best impact we have on future generations. But teaching is also incredibly hard. Most computer science professors are never formally trained (much of the field is still new and changing), so they often rely on their own experiences of being taught and attempting to teach. While I believe teaching experience is necessary for becoming a good teacher, a strong teaching philosophy is essential for becoming a great teacher.

TEACHING EXPERIENCE

The experiences that have made the most impact to me, personally, have been the 2.5 years as a senior teaching assistant for a sophomore computer science class involving hundreds of students (CS 2130: Languages and Translation). The class taught C programming, with the additional topics of regular expressions, formal grammars, and building lexers and parsers (by-hand and with tools: lex/yacc). The professor taught lectures, but much of the teaching responsibility was handled by the TAs. We taught a 1.5 hour recitation and held a 1.5 hour lab weekly. In addition, we created and graded homework assignments, projects, timed lab assignments, and tests; answered newsgroup questions; and held office hours. We also had to deal with plenty of stressed and confused students who feared flunking out of their major.

TEACHING PHILOSOPHY

As a result of my teaching experiences, I have come to understand the many facets and skills needed in education: managing and delegating the multiple duties involved in running a large course, navigating the gray areas of grades and evaluation, distilling and relating difficult concepts, engaging and motivating students, and developing course content.

My teaching philosophy follows many of the tenets of transformational teaching, a teaching perspective where the goal is not only to successful transmit critical skills and knowledge, but also to create self-directed and self-motivated learners through the development of skills such as inquiry, critical thinking, and communication.

I believe teaching should involve engaging topics that are explained with fundamental concepts. A classroom should facilitate peer learning and interleave hand-on work with lectures. Having worked full-time professionally throughout my entire education, I believe getting exposure to realistic situations and environments is essential in learning how to build software. I believe in cultivating a learning atmosphere that encourages self-learning without artificial and arbitrary constraints. To promote self-learning, I will use a model that allows students to choose from multiple learning units in the course completed within their own timeframe.
TRANSFORMATIONS IN SOFTWARE ENGINEERING EDUCATION

Teaching experience and a strong philosophy is only part of what is needed to successfully engage and teach students in software engineering.

EMPIRICAL FOUNDATIONS OF SOFTWARE ENGINEERING

In order to properly teach software engineering, we need to derive practices for building software from empirically validated scientific principles. Although we have plenty of principles and practices, we have little evidence that validates them. As Greg Wilson notes, when teaching other scientists the “best practices” of software engineering, “scientists know good evidence, and they knew software engineering has none”. When teaching software engineering curriculum, I believe it is essential for students to understand and debate the evidence behind a prescribed practice. For practices with limited evidence, students should learn how to design studies that can validate them. As much of my research experience relates to the empirical side of software engineering, I will have a good handle on teaching students our current evidence as well as enabling students to learn how to debate and validate existing practices.

EDUCATIONAL TECHNOLOGIES: LIVE PROGRAMMING

Students still have incredible difficulty in learning programming concepts, which often detract from other learning goals. In particular, I believe live programming environments, which allow students to acquire rapid feedback when coding, offer an promising way of improving how students learning programming. As a recent co-organizer of the first international LIVE programming workshop and a panelist on live programming at VL/HCC, I have seen many exciting approaches and systems being built that I believe can be incorporated into the classroom. Not only is this a way for students to watch a lecture and see algorithms unfold in a real-time manner, it has the opportunity to make homework and project assignments less frustrating and more intuitive. I am excited for the chance to introduce new educational tools in teaching computing.

MY FUTURE COURSES

SOFTWARE ENGINEERING/PROGRAMMING LANGUAGES COURSES

Based on my industrial experience and prior teaching experience, I am excited to be able to teach undergrad and graduate computer science courses, especially those related to software engineering and programming languages. In teaching software engineering and relating it to real-world practices, I can draw from a wide range of professional experiences collected over a decade of software development in the defense industry. Some example topics include processes, teams, workplace factors; mocking, refactoring, testing; architectures, frameworks, design patterns; and evidence-based practice of software engineering.

During a lecture, students break out in small groups and discuss with their group their own experiences or thoughts on a class topic, and then share their discussion with the class. During class assignments, students get hands on experience with several practices through activities such as by coding in pair programming teams and performing code reviews. Finally, students will be well-prepared to use state-of-the-art software engineering tools in the context of professional practice: agile trackers, distributed version control, social coding sites (e.g. github), online code review tools, and testing tools.

Finally, a student can take on one of many roles in the software industry. For example, typical roles include architects, automated testers, back-end developers, build engineers, or UX designers. To facilitate understanding and gaining specific training in those roles, students will have the opportunity to select from multiple learning units and assignments in the course that allow students to gain proficiency in roles of their interest.

ADDITIONAL COURSES

In addition to the traditional material just described, I describe two additional courses that I would love to develop and teach. While I realize depending on the college’s current program that additional courses may not be needed, the example courses illustrate how I would teach senior and graduate level topics in areas that intersect computer science and software engineering. Finally, there is good reason to believe that these types of courses may be needed some day.
Neuroscience Programming

Neuroscience is an exploding field; not only does it involve scientific computing (medical imaging, neural-controlled prosthetics), but computational tasks are also increasingly involving neuroscience (computational models based on neural systems, and brain-computing interfaces). In the class, students will learn elementary cognitive neuroscience concepts and acquire the skills necessary for processing data and building applications related to neuroscience. Students will work with existing datasets collected from biometric sensors and brain imaging scans as well as collect and process new data in labs. Finally, students will have a chance to build applications based on brain-computing interfaces. Example assignments include detecting words from subvocal utterances, detecting anxiety from galvanic skin responses, and using functional near-infrared spectroscopy (fNIR) to control inputs into a computer game.

Crowd Systems

Many emerging programming tasks revolve around delegating computational tasks to crowds of workers (For example, this document was proofread via Mechanical Turk using http://proofist.com/). Companies are also interested in designing and building custom systems that support crowd-level interactions for their specialized domains. For example, one company consulted with me on a system allowing for “a crowd of external members to make comments on proposed changes to national fire safety code documents.” In the class, students will learn about principles of crowd-sourcing and large-scale social systems. They program tasks that work on existing sources (e.g. Mechanical Turk) as well as design and build new types of crowd systems. Finally, the nature of this course is suitable for teaching in both a traditional and MOOC format.

Mentoring Students

Recently, I have had the experience of co-advising a masters student on his thesis work. Additionally, I have mentored several students and even other professors on running user studies on software engineering tools, including serving as an expert at usability study workshops. For me, the primary challenge of a being mentor is in providing just enough guidance on problems, but not to the point of outright solving them. Delivering a monologue instead of engaging the learner is an easy trap to get caught up in.

Although my mentoring philosophy is very similar to my teaching philosophy, when mentoring, I believe in additionally incorporating several of the tenets of developmental teaching. From a developmental teaching perspective, I believe in supporting students in bolstering their weaknesses and honing their strengths in a manner best suited for their ultimate career goals. For example, in my experience, some students can make significant progress based on high-level directions or tasks, whereas others will just flounder unless they are given very specific directions and small tasks. In the later case, I would additionally focus on developing the student’s task planning skills while still giving specific directions until they are ready to do so on their own. From a transformational teaching perspective, when mentoring senior and graduate students, if I could teach a student three skills, it would be: how to self-learn, how to read a research paper, and how to make an argument that is supported by evidence.

Finally, as I have done when I was a teaching assistant, I will always make myself fully available to any student who needs mentoring not only on class topics but on personal and professional goals.

Conclusion

My experience and passion for teaching will ensure that will students excel in their education. New faculty in computer science may be overseeing a period of transformation in both the discipline itself and within the centuries old institute it is held. My experience in empirical software engineering and educational technology will contribute to growing a stronger software engineering program within my future department. Finally, I am excited to take leadership in adapting computer science education in alignment with the educational goals of my future department.