Teaching Statement

Gowtham Kaki

In a staunch defense of his life's work in pure mathematics, Prof. G H Hardy writes in *A Mathematician's Apology* that the "real" mathematics is almost wholly "useless" whereas useful mathematics is "intolerably dull". A real mathematician therefore ought to indulge in abstract mathematics, which in Hardy's opinion has the most aesthetic value albeit being devoid of any worldly purpose. A physicist's real world, Hardy contests, is merely a model of the mathematician's reality, which is more general and all-encompassing. The pure mathematician therefore need not be constrained by physical facts, especially because, as Hardy argues, "imaginary universes are so much more beautiful than this stupidly constructed real one". Hardy's unapologetic disdain for any materialistic application of mathematics is patently evident in his book *A Course of Pure Mathematics*, where the reader is hard-pressed to find a single demonstration involving a real-world circumstance.

Teaching Philosophy

Research in Computer Science is not very much unlike the research in mathematics in that it too involves building and reasoning about elegant abstractions, such as the lambda calculi and the petri nets. It is therefore not inconceivable for us to sympathize with Hardy, and de-emphasize the real-world utility of our abstractions in favor of the study of the abstractions themselves. However, I believe that our sympathies with Hardy must lie strictly within the confines of our own research and not permeate our teaching, for I believe it is extremely counterproductive to teach an abstract concept untethered from its real-world motivation. This leads me to the first tenet of my teaching philosophy: *utility over aesthetics; problems over solutions*. In my observation, the common motivation for students to enroll in a course is to acquire the skills that they believe will help them advance their career. It is therefore understandable that if a student fails to see a clear causal relationship between a concept taught in the class and her professional success, she would disengage from the learning process, notwithstanding the constant exhortations about the aesthetic value of the concept. It is therefore imperative that a student sees a compelling utility of what is taught in the class in the context of her professional journey – a problem to which she *has to* know the solution, before she is presented with a solution and the overarching concept. Such problem- and utility-guided approach to teaching is, I believe, more productive than the alternative approaches that stress on solutions, concepts and their aesthetics.

The second major tenet of my teaching philosophy is to prioritize *clarity over truth*, i.e., to sacrifice details, or even *lie to students* [2], if it helps them develop intuition and clear mental models for complex (computational) phenomena. The mental models thus developed may be incomplete, or even (partially) incorrect, but insofar as they help students internalize and reason about a *subset* of the complex phenomena correctly, I would consider them useful. It is preferable to gradually refine coarse intuitions and build on simple-but-incomplete mental models, rather than to commit to a hard-to-digest truth right from the outset and confuse students. This opinion is admittedly controversial, yet I am not alone in harboring it for I find support from influential thinkers and teachers who share similar sentiments [1, 2, 4].

Note that de-emphasizing truth in favor of clarity does not give one the license to hand-wave the details and be lax in the presentation; quite the opposite in fact. The third tenet of my teaching philosophy is the belief that *rigor begets clarity*. A case in the point is a mathematical proof. When one finds a proof hard to understand, it is likely the case that it skips supposedly trivial details that turn out to be non-trivial, and makes intellectual leaps that are too large for one's mind to bridge. In other words, the proof lacks rigor. Similar case can be made of concepts discussed regularly in CS classrooms. A CS instructor may prioritize comprehension over correctness and choose to not discuss an algorithm in its full generality, but I believe that the purpose of comprehension is not met unless the simplified algorithm is discussed with a sense of rigor appropriate for the class. In other words, rigor is a necessary precondition for clarity.

The fourth and the final tenet of my teaching philosophy is the belief that *inclusion should not be an afterthought*. Thanks to the K-12 outreach of CS departments around the world, the enrollment of underrepresented groups in CS is steadily increasing. While the outreach efforts can *bring* students from disadvantaged sections to the classroom, I believe it is the responsibility of a CS instructor to *keep* them in the classroom by making them feel included. Fortunately, commitment to inclusion does not necessarily entail an overhaul of one's teaching methodology; simple gestures often make a significant difference. For instance, choosing examples carefully to avoid propagating the stereotype threat in classrooms has been known to be quite effective [6, 7]. Another simple gesture is to consciously refrain from speech that is implicitly judgmental and promotes *defensive social climate* [5]. For instance, the declaration "smart students who finish early will receive extra credit" carries with it the connotation that students who finish on time or late are not smart. Declarations like these must be avoided. It is by combining such simple gestures with the rigors of teaching CS that I believe we can make CS classrooms more inclusive and diverse.

Teaching Experience

I am fortunate to have gotten the opportunity to serve as Visiting Assistant Professor at Purdue since August, 2019. For the current (Fall'19) semester, I am teaching CS307 Software Engineering to undergraduates in their junior year, which gives me an opportunity to implement some of my teaching ideals in practice. The course aims to equip students with various skills that help them organize a software development project, and deliver reliable software in time and within budget. The curriculum includes such things as Agile principles, manual and automated software testing, symbolic modelchecking, and code review. The course is primarily taken by students who intend to specialize in software engineering, hence the pedagogical and learning goals are in perfect alignment. It was therefore not hard for me to organize my teaching around the principle *utility over aesthetics; problems over solutions*. For every topic I teach, I give a sufficiently elaborate demonstration of the utility of the topic in real-world software development with an emphasis on the problems it aims to solve. It also helps that students work on a semester-long software development project, which acts as a familiar common context for effective demonstration of software engineering concepts. Advanced software testing and modelchecking algorithms are taught by first simplifying them to capture their essence (thus prioritizing *clarity over truth*), and running through the simplified algorithm with help of several examples (i.e., with sufficient *rigor*). Due attention is paid to social aspects of teaching whenever an opportunity presents itself. For instance, when an advanced testing technique is discussed, I consciously acknowledge the difficulty in grasping the technique so that the students who did not understand it do not feel invalidated. Another regular practice is to use the pronouns "she" and "her" to address a programmer so as to counter gender stereotypes prevalent in tech.

At the time of writing this statement, Fall semester is ongoing and the class is still in session. However, the feedback I got so far, directly and indirectly, is quite encouraging. At the end of a particularly engaging recent lecture on concurrency bugs, a student excitedly walked up to me to let me know that he is thoroughly enjoying the lectures. An indirect encouragement I receive from students is in form of the high attendance to my lectures even on days when there is guaranteed to be no quiz in class. I plan to build on this encouragement in future, as I look forward to challenging teaching assignments in Programming Languages, Systems, and Software Engineering.

Mentorship

During my graduate studies at Purdue, I have had the opportunity to mentor several junior graduate students, few of whom have later became my research collaborators. It is a matter of pride to me that on at least four of my research papers, my immediate co-author (2nd or 1st) is a junior graduate student for whom it was the first ever publication. I am also comfortable mentoring senior researchers who are more accomplished than I when they choose to expand into a research area that is more familiar to me. On one of my papers, my immediate co-author was a post-doctoral researcher for whom it was the first ever research paper in programming languages. He has since begun his own line of exploration, and has become a regular contributor at PL venues. Beyond one-to-one mentorship, I have also taken initiatives for collective professional advancement that have stood the test of time. PurPL, a PL reading group I started in Fall 2013, has since branched off and matured into a full-fledged research organization[1]. I would love to take more such initiatives in the future for nothing gives me more joy than sharing the excitement of learning and discovery in Computer Science with my fellow researchers.

References

[1] Wittgenstein's Ladder, 2019. URL `https://en.wikipedia.org/wiki/Wittgenstein%27s_ladder`.

[2] Lie-to-Children, 2019. URL `https://en.wikipedia.org/wiki/Lie-to-children`.

[3] Purdue University Programming Languages Group, 2019. URL `https://purduepl.github.io`.

[4] Teaching Tech Together, 2019. URL `http://teachtogether.tech/#the-rules`. How to create and deliver lessons that work and build a teaching community around them.

[5] Lecia Jane Barker, Kathy Garvin-Doxas, and Michele Jackson. Defensive climate in the computer science classroom. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, pages 43–47, New York, NY, USA, 2002. ACM. ISBN 1-58113-473-8. doi: 10.1145/563340.563354. URL `http://doi.acm.org/10.1145/563340.563354`.

[1] PurPL, as it is known now [3], is largely due to the efforts of Profs. Tiark Rompf, Milind Kulkarni, Roopsha Samanta, and Ben Delaware. The reading group continues nonetheless.

[6] Matthew S. McGlone and Joshua Aronson. Forewarning and forearming stereotype-threatened students. *Communication Education*, 56(2):119–133, 2007. doi: 10.1080/03634520601158681. URL `https://doi.org/10.1080/03634520601158681`.

[7] Steve Stroessner and Catherine Good. Stereotype threat: An overview. 2011. URL `https://diversity.arizona.edu/sites/default/files/stereotype_threat_overview.pdf`.