Teaching Statement

James R. Wilcox

I view the challenge of developing and teaching a class as a *design* problem, where the student is viewed an active participant in the resulting system, whose experience must be carefully considered and centered during the design process. Every student is a unique individual who brings different skills and needs into the classroom. My experience in course design has led me to develop an approach that emphasizes tight feedback loops, places grades as subservient to feedback, and invites students to participate as active stewards of our shared intellectual heritage. Through these techniques, I aim to develop inclusive courses that invite all students into a community of learners that welcomes and supports them as they grow to carry our field into the next generation.

Designing Scalable, Interactive, and Adaptive Feedback Loops

In addition to being a computer scientist, I am also a student airplane pilot. When controlling either a computer or an airplane, feedback is essential. I cannot debug a problem if I have no information beyond "something is wrong", and I cannot fly an airplane without information about its orientation in space. As a teacher, I also know that feedback is essential to the student's learning experience. Students need to know where they are succeeding and where they are not in order to allocate their effort wisely. These two forms of feedback are related but distinct, and I believe good course design carefully considers them both, and their interactions. For example, if a student comes to me for help with a homework question, not only do I want to give them feedback on their current attempt, but I also want to give them feedback on their strategy, and to teach them where to look to get the feedback they need from the system. From the perspective of a student's entire experience in a course, they will get some feedback from course staff, and perhaps some from their peers, but they will get far more feedback from the tools and systems we ask them to use and interact with, because these tools and systems are inherently more scalable than finite course staff resources. As a result, selecting or designing tools that provide appropriate interactive and adaptive feedback, and explicitly teaching students how to use these tools effectively is essential to student success.

When I teach at the 300 level, aimed at a wide range of undergraduate CS majors or non-majors, I do a significant amount of live coding in front of students. I try to show them not just how to write the solution, but how they can derive the solution by working with the course tools effectively. I like to tell students that it's important to know when to turn your brain on and when to turn it off. Some problems are best left to tools rather than our brains. In the undergraduate programming languages class (CSE 341) at the University of Washington, we study both statically (compile-time checked) typed and dynamically (run-time checked) typed languages. In a statically typed language, the compiler will *guarantee* that our program doesn't accidentally use a number where a list was expected, or that we don't refer to a field that doesn't exist. When using this tool, we can safely turn that part of our brain off and trust in its feedback. For example, if we want to refactor a program in such a language to remove a field from a class, we don't need to worry about remembering exactly where all the references to that field are. Instead, just run the compiler, and fix all the errors it reports. As another example, in CSE 374 (non-majors introduction to systems), I emphasize liberal use of additional flags to the C compiler that turn on more warnings and enable sanitizers.

I don't draw a strong distinction between designing course materials and designing tools. Sometimes, the best way to teach a topic is to build a tool that let's students explore it. For example, when teaching CSE 490P (advanced undergraduate programming languages) and CSEP 505 (graduate programming languages), I found that students struggled with some of the theoretical languages that we study. These languages are unlike mainstream programming languages in that they are defined by a pen-and-paper description, and they are not executable, so students cannot try out example programs themselves without a great deal of cognitive overhead of churning through the pen-and-paper rules. To improve their experience, I either have students implement the languages themselves (490P), or I provide such an implementation for them (505, see `https://jamesrwilcox.com/f/`). Then I explicitly ask students to write a few programs in these languages and *interact* with these tools, which I have found helps students get comfortable with the concepts; it also relates the theory to students' previous programming experience more directly, and makes it more fun.

Grades are Second to Feedback

While interactive feedback from tools helps students on a minute-by-minute basis with their work, feedback from course staff and course peers at a coarser granularity helps guide students in developing their own sense of taste in terms of well written code or proofs. Traditionally, this coarse course feedback is communicated by, or at the very least conflated with, grades. I have been lucky to participate in many conversations about grading with my colleagues at the University of Washington, especially in CSE 590E, the computing education research seminar. Out of these conversations, I have experimented with several alternative grading strategies, with goal of moving grades out of the way of giving effective feedback.

When I taught CSE 374 in winter quarter 2021, I piloted a contract-based grading scheme, where each assignment asked students to write a program that exercised certain features of the C or Bash language that we were studying that week. We organized both peer and staff feedback on these assignments. All feedback was conducted through code reviews using Gitlab, and assessed solutions according to their understandability and whether they correctly used the desired features. From a grading perspective, I marked students essentially on a completion basis. Even though I told students that *any* program that met the contract would receive full credit, I was blown away by the creativity and effort that many students put into their programs. For example, several students implemented various kinds of games, some of them graphical, even though they had no prior experience with C, and the class did not cover graphics at all. On the other hand, an important hazard with this approach is that some students do not like to feel “forced” to be creative. To mitigate this, with each assignment, I also provided a few optional ideas for programs that they could write that would allow them to meet the contract.

Another instance of moving grades out of the way of feedback was my experience co-teaching CSE 505 and CSEP 505 with Zach Tatlock. We piloted an additive, points-based grading system. At the beginning of the quarter, we published exactly how many assignments of various kinds there would be, and how many points they would each be worth. We also published a formula for computing a 4.0-scale grade from the total points a student received over the whole quarter. We explicitly told students that they did not have to do all the assignments. They could select their desired target grade and decide which assignments, and how much of each, they wanted to complete. This gave students agency over their grade and course experience, and many students reported they appreciated the grading system in the course evaluations.

Active Intellectual Stewardship

I have always viewed teaching as intimately connected with research. In 2018, I applied to a few tenure-track positions before deciding that that route was not for me, and in my teaching statement at that time, I wrote that teaching is an integral part of my role as a scientist and researcher. Since then, I have gained a significant amount of additional teaching experience, and I now feel the same way as I did then, but from the other side. I view my ongoing participation in the programming languages and systems research communities as an important aspect of my teaching approach. I consider myself incredibly lucky to be able to learn and teach ideas from such a rich field, and it is my goal to pass on the best of these ideas to my students, and to support them participating in the community. This is a form of *intellectual stewardship*, in which I view my role as a maintainer and promulgator of these beautiful ideas.

My background is in programming languages (PL), and one of my favorite aspects of PL is how good theory leads to good code. For example, in CSE 490P (and, if I were to teach it at some point, CSE 401, compilers), an important learning outcome is the ability to read what are called “inference rules.” These are a somewhat intimidating notation that is widely used in the PL community to write down inductive definitions. They are especially common when writing down type systems, which have a very deep theory. Through my participation in the PL research community, I have been lucky enough to learn a great deal of this theory, and I believe it can be made accessible to undergraduates in a way that benefits them. For example, one can specify the type system for the MiniJava language used in CSE 401 in just a page or two of inference rules. When students learn to read and fully understand this notation, implementing a typechecker in their compiler becomes “just” a matter of translating the inference rules to code.

More generally, I view the history of ideas as a sequence of steps that filters and polishes the best ideas in order to pass them on to wider and wider audiences. I try to invite students to join me in the process of polishing these ideas by phrasing them in their own terms. Not a quarter goes by that I don't learn something more deeply myself because of a question a student asks, or from a connection they draw, or from the way they rephrase an idea. The resulting learning from and with students, not just teaching at them, is an aspect of my experience that I cherish.

Another aspect of stewardship is mentoring in both teaching and research. As a grad student, I had the pleasure of working with 7 undergraduate student researchers on various projects. I continue this aspect of my mentoring now through the SIGPLAN-M long-term mentoring program, in which I mentor 5 junior Ph.D. students from around the globe. As an instructor, I have worked with over 30 (primarily undergraduate) TAs, many of whom have gone on to graduate work in CS. I also consider myself a mentor to students in my classes. I am always surprised, for example, how many CS majors at the University of Washington don't know that Ph.D.s in computer science are fully funded, and I make it a point to mention this in every undergraduate class I teach.

My experiences as an instructor have been challenging but incredibly rewarding. By viewing course design *as design*, with the student experience centered, I strive to help students find the feedback they need, to move grading out of the way of good course feedback, and to walk with students on our journey as intellectual stewards of the ideas we are lucky to explore together.