Overview

In my classes, I strive to create an inclusive learning environment that allows all students to develop a strong foundation in computing. At the undergraduate level, I teach a sophomore-level programming course that is required for all computer science majors. I completely redesigned the course to teach functional programming and advanced object-oriented design. At the graduate level, I designed a new programming languages “core” course that covers a variety of topics on the theory and practice of programming languages. I’ve also taught and co-taught graduate seminars on several topics, e.g., formal verification with the Coq Proof Assistant.

Undergraduate Teaching

I teach a sophomore-level course that focuses on the development of programming and software engineering skills. The course is at a critical junction for students: most enter the course with only one year of Java programming experience and then leave to take electives that assume programming proficiency. In my view, the goals of the course are threefold: (1) strengthen their understanding of programming foundations, (2) drill them to develop strong programming skills, and (3) give them the confidence and ability to learn new programming languages, APIs, and technologies on their own. The latter objective is especially important to help students keep up with rapidly-changing technologies.

When I first taught the course, I made several significant changes to the curriculum:

- *Programming from scratch.* Given a high-level problem specification, novice programmers often suffer from “the blank page syndrome” and cannot get started. I emphasize a programming methodology where students start by formulating their own examples (i.e., tests), breaking the problem into pieces (i.e., signatures for helper functions), and only then write actual code. I emphasize the relationship between the structure of datatypes and the structure of code and explicitly teach design recipes [2]. Most assignments in the course have students write programs from scratch with no support code whatsoever. Many assignments have students write their own data structures that aren’t part of the standard library.

- *Mandatory, graded test suites.* I require students to test their code and grade their test cases in several assignments. Without thorough testing, students cannot get full credit even if their programs appear to run correctly on my test cases.

<table>
<thead>
<tr>
<th>Course Code</th>
<th>Course Title</th>
<th>Terms Taught</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPSCI 220</td>
<td>Programming Methodology</td>
<td>2014F (144 students), 2016S (161 students), 2016F (163 students), 2018S (174 students)</td>
</tr>
<tr>
<td>COMPSCI 631</td>
<td>Graduate Programming Languages</td>
<td>2013F (13 students), 2016S (21 students), 2016F (15 students), 2017F (25 students)</td>
</tr>
<tr>
<td>COMPSCI 691PL</td>
<td>Seminar: Advanced Programming Languages</td>
<td>2014S (8 students), 2016F (7 students)</td>
</tr>
<tr>
<td>COMPSCI 697NN</td>
<td>Seminar: Topics in Programming Languages, Software Engineering, and Systems</td>
<td>2017F (8 students)</td>
</tr>
</tbody>
</table>

Figure 1: Summary of teaching at UMass Amherst
• Modern programming principles. I introduced Scala as the language of the course to emphasize both object-oriented and functional programming principles. Functional programming ideas are found in several popular, modern programming languages, such as JavaScript, Python, C#, and even Java 8. By introducing students to these ideas early, they emerge better prepared for today’s computing jobs.

The new curriculum appears to be popular with students. In an end-of-class survey that I conducted in Fall 2014, 83% of students said they preferred Scala to Java with 70 of 144 students responding.

A potential pitfall in a lower-level undergraduate course is that it can be overwhelming for novices and only accessible to students who enter college with prior programming experience. Conversely, a course can also be too easy for advanced students. I wanted to avoid these failure modes and surveyed students who received an A grade to find out how much time people spend on the course and how much prior programming experience they have. The survey indicated no correlation between these factors.

Graduate Teaching

My graduate programming languages course is a core course (i.e., satisfies requirements for PhD candidacy) that covers the theory and practice of programming languages. My broad goals are (1) to show students the elegant and rigorous mathematical foundations of programming languages, (2) to train them to apply this rigor in the implementation of language-based tools, and (3) to show that programming language principles can be applied to a variety of domains. The last objective conveniently ties the course to my own research, which applies programming language technology to security, networking, and systems.

My course leverages material from similar courses at Brown University and Cornell University [3 4], where I was a PhD student and postdoctoral researcher. I also cover recent topics in language-based web security and network programming languages. Not only are these my research areas, but they do a nice job of showing students that programming language principles are widely applicable.

I have taught and co-taught graduate seminars in several topics. For example, my first seminar covered mechanized theorem proving using Certified Programming with Dependent Types [1]. After a few weeks of lectures and homeworks, students used Coq for a variety of their own projects, from modeling type systems to verifying Web-authorization protocols. My most recent seminar had students reproduce results from recent programming languages papers.

Advising

Presently, my research group consists of three PhD students and one undergraduate student. I have supervised undergraduate research with students from my university and several other institutions (Mount Holyoke College, Grinnell College, and IIT-Madras). I have graduated one masters student and several undergraduate students. Three of my undergraduates have gone on to PhD programs: Akshay Gadre (Carnegie Mellon for networking), Aaron Weiss (Northeastern for programming languages), and Rachit Nigam (Cornell for programming languages).

References


