Lucas Layman
9/16/2007

Teaching Evaluation Reflections

The amount of information I could gather with regards to my particular teaching was somewhat limited. I was responsible for teaching an active two-hour lab session during the semester of a project-intensive course in Software Engineering, but I was not registered as an instructor as such, and therefore didn't receive my own evaluations. Instead, the TAs for this course were evaluated on with the general course, and most of the feedback from students came from free response questions on the course evaluation. Since the TAs were very extremely involved in conducting instruction and creating homework assignments and the course project, I was able to glean important considerations for teaching styles and effective student engagement from the students' general comments on the course.

Software engineering is a junior-senior level course, and most courses have focused strictly on theory and fundamentals of computer science, rather than practical application. The course has two, one-hour lectures each week, plus a two-hour, closed lab wherein the students practice tools and techniques for software engineering, work on homework assignments, and work on the group project, all under the close supervision of the TA. Indeed, student evaluations indicate that the majority of the learning occurs in the lab. As the comments bear out, one of the leading reasons that the students prefer the labs is because of the practical nature of what is taught. The students perform hands-on learning, are free to collaborate with one another, and can discuss ideas with the TA. The comments reflect a sometimes forgotten idea: that many people do not learn concepts well alone (as in a lecture) without seeing the concepts in action and experimenting with them. As an instructor, teaching the lab is also quite rewarding, despite requiring more preparation than may be customary for a lab. When the students are engaged and collaborating with one another to solve technical problems, the instructor takes a back seat as more of an observer and consultant rather than a pedagogue.

In the software engineering course, one of the main instructional goals is to provide the students with a sense of what it is like to work on a "real" software project. Achieving realism in the course has proven to be quite difficult. Our main approach is two-fold: 1) for smaller assignments, give them a real-world context, and 2) find a real, external customer for the course project (as opposed to a contrived project on the part of the teaching staff). While we typically have no difficulty providing real-life context for small assignments or finding an external sponsor for the course project, preparing the assignments and the project require an enormous amount of effort on the teaching staff. This is primarily because the students typically do not have all of the necessary technical skills to create an entire, working software project, which requires a tremendous amount of infrastructure and technical support. Thus, in our quest for realism and engagement, we place so many learning requirements that sometimes learning goals are diluted or lost. In short, the students can't see the forest for the trees.

Software technologies have sharp learning curves, and the students typically have to learn several to complete their project. Also, creating a software project requires a tremendous amount of programming, which can be a menial task at times, and takes away from more important learning goals, such as good software design. Thus, we are faced with a difficult challenge: how do we maintain realism and engagement without overloading the student? We tried to address this problem in a later semester by providing the students with a significant amount of code and infrastructure up-front. Initially, this confused the students, and they fell into a mode of "just tell me what to do to make it work with your stuff." In recent semesters, we have provided the students with a software project that was created for internal purposes (no external customer), and we were able to structure the project much more controllably, thereby providing the students with a more learnable environment. However, the project also lacks the total realism of an external project. At present, it appears that the best solution would be a tradeoff between total realism and a less intensive, more-controlled project that can be more closely aligned with instructional goals.

Aside from the project and course content, the evaluations provide a number of useful reflections on teaching style. Students desired clear goals and expectations for materials they will be evaluated on. Being a student is time-intensive, and the software engineering course is the most time-intensive course in the software engineering curriculum, and many students complained of the work load. Some students expressed a desire to have learning goals and evaluations clearly laid out so that they can focus their learning efforts in light of a tremendous amount of work. Clearly, reducing the amount of work required would help as well, though the requirements for material to be covered in the course are strict and numerous.

One of the most common themes among the student evaluations regarded the personal relationship between instructors and students. We show that we care about the students, that we want them to really *learn* the material, and that we want them to succeed. We try to be enthusiastic, we get to know our students, and we are responsive to their email and message board questions. The students very much notice and respond to this effort (and they notice a lack of such effort). They are not afraid to ask questions, they are not afraid to voice concerns, and they are willing to learn if you are willing to help them do so.

Clearly, all of the above issues are important. Students want to learn and they want to be engaged. We can teach effectively and provide a fantastic learning environment, but the scope of the material we are trying to teach is so broad that it has become a real problem. I am not sure how to deal with this. In the large, the problems facing the software engineering course seem to be a curriculum problem in general, in that the students are required to learn so much in a single semester, but also they seem to be lacking in certain fundamental skills when they arrive in the course and thus must spend time "catching up." There are currently discussions within the department to expand software engineering to be a two semester course, lightening the burden on both students and instructors.