

Teaching Statement

Lucas Layman

A teacher is a *facilitator of student enablement*. Enablement takes several forms. *Instruction and exposure* provides students with the knowledge and tools to solve problems. *Fostering interest* connects the computer science domain to the students' inner motivations to demonstrate how computer science will help them achieve their life goals and, hopefully, will create a lifelong passion for the discipline. Finally, students should learn *how to teach themselves* so that they are prepared to meet any challenge through their own autodidactic powers. A necessary component of achieving these teaching goals is to strive for a universal design for engaging all students regardless of gender, ethnicity, age, personality type, learning style, or motivation.

Most of my teaching experiences originate from serving as a teaching assistant (TA) for four semesters in the undergraduate software engineering class. The software engineering course is project-based and typically consists of 50-60 junior- and senior-level students. The students are required to attend two 50-minute lectures and one 2-hour closed lab each week. During my time as a TA, I was fortunate to have a great deal of influence on both the structure and administration of the course. I enjoy teaching software engineering, and feel comfortable with related topics such as software testing, object-oriented design, and would be willing to teach foundational topics such as introductory programming.

Active learning is integral to achieving my teaching goals and to preparing students for professional careers. In our course, students receive hands-on experience in the weekly lab session and use technologies such as the Eclipse integrated development environment (IDE), the JUnit and FIT testing frameworks, UML diagrams, version control software, and object-oriented design patterns that are used by practitioners. Active learning allows students to see concepts at work, to test the boundaries of these concepts, and to experience first-hand the utility and applicability of software engineering tools and techniques. For example, students sometimes have difficulty understanding the concepts of equivalence-class partitioning and boundary-value analysis testing strategies – admittedly heady terms. Yet, these terms are quickly illuminated in lab sessions when the students begin testing user input or array index limits. In this “learning-by-doing” approach, the students also are teaching themselves by discovering questions and answers on their own. Furthermore, active learning appeals to many students who have a personality type or learning style that is conducive to hands-on learning [1, 3].

Another core component of achieving my teaching goals is student collaboration. Students should collaborate when learning new software engineering skills in the laboratory, when applying those skills to solve programming assignments, and when creating a software system as part of a team project. Collaboration is a powerful tool demonstrating that students can resolve problems and learn without instruction from “on high” [4-6]. Students rely on each other for technical knowledge and complement their understanding of a concept with multiple viewpoints [4, 6]. Collaboration also demonstrates that software development, contrary to popular misconception, is a social activity rather than a solitary one. Thus, collaboration can be a mechanism to appeal to more socially-oriented groups, such as women and

ethnic minorities [3]. Through collaboration, students also gain experience communicating their thoughts to others, and solidify their knowledge through peer dialogue.

A final component to achieving my teaching goals is creating practical and socially-relevant activities. Practical, socially-relevant, and even fun assignments serve to illustrate points but also help the students to understand the capabilities and application of their chosen field [2]. Such assignments also help to answer that common question, "Why am I doing this?" through demonstration rather than through explanation by the instructor. Also, research has suggested that social and practical considerations appeal to women, ethnic minorities, and the Millennial generation [2, 3]. In our class, the team project has taken the form of a defect reporting tool, an online database for a state forest, and a healthcare application. Students' evaluations reflect an appreciation for more "realistic" assignments over straightforward problem-solving exercises.

I evaluate whether I have met my teaching goals in a number of ways. Students should be able to apply software engineering tools and techniques to homework assignments and to the class project. On individual exams, students should also be able to recall information, apply their knowledge to theoretical scenarios, and identify and discuss the pros and cons of their solutions to analytical problems. If I have successfully fostered an autodidactic nature in the students, their questions in lecture and in lab are more conceptual and theoretical rather than the rote technical questions that can be answered readily by their peers or other resources. I know I have successfully encouraged interest in the material when students ask if they can implement "neat" features in their projects for extra credit, or when they bring me stories from news media or from their jobs relevant to software engineering topics. During the course, frequent feedback (anonymous and direct) on both content and instruction is one of the best ways to improve the educational experience of students. A combination of the aforementioned evaluation techniques, iteratively reviewed, provides the best formative evaluation for ensuring that students are learning the course content, are interested in the software engineering discipline, and are becoming self-directed learners.

References

- [1] L. Layman, T. Cornwell, and L. Williams, "Personality Types, Learning Styles, and an Agile Approach to Software Engineering Education," in *ACM Technical Symposium on Computer Science Education (SIGCSE '06)*, Houston, TX, 2006, pp. 428-432.
- [2] L. Layman, L. Williams, and K. Slaten, "Note to Self: Make Assignments Meaningful," in *ACM Technical Symposium on Computer Science Education (SIGCSE '07)*, Covington, KY, 2007, pp. 459-463.
- [3] L. Layman, L. Williams, K. Slaten, S. Berenson, and M. Vouk, "Addressing Diverse Needs through a Balance of Agile and Plan-driven Software Development Methodologies in the Core Software Engineering Course," *International Journal of Engineering Education*, in press.
- [4] K. M. Slaten, M. Droujkova, S. Berenson, L. Williams, and L. Layman, "Undergraduate Student Perceptions of Pair Programming and Agile Software Methodologies: Verifying a Model of Social Interaction," in *Agile 2005*, Denver, CO, 2005, pp. 323-330.
- [5] L. Williams and L. Layman, "Lab Partners: If They're Good Enough for the Sciences, Why Aren't They Good Enough for Us?," in *20th Conference on Software Engineering Education and Training*, Dublin, Ireland, 2007, pp. 72-82.
- [6] L. Williams, L. Layman, K. M. Slaten, C. Seaman, and S. B. Berenson, "On the Impact of a Collaborative Pedagogy on African-American Millennial Students in Software Engineering," in *International Conference on Software Engineering (ICSE '07)*, Minneapolis, MN, 2007, pp. 677-687.