Teaching Statement
Scott D. Fleming

As a CS educator with roughly a decade of experience, I am passionate about preparing graduates for careers in computing and software engineering (SE). According to the ACM/IEEE CS Curriculum Guidelines (CS2013), the expected outcomes for SE students go far beyond the ability to program software. Graduates must master the key SE principles, such as modularity and abstraction, as well as the heuristic methods and best practices, such as software design patterns, which support those principles. Moreover, they must acquire the practical skills required for success in a real-world development environment, such as the ability to work effectively in teams and to use modern development platforms and frameworks. It is not sufficient for student to be familiar with the methods and tools for solving these problems, but they must be able to apply SE knowledge in new and unfamiliar situations.

1 Real-World Tools and Technologies

A key feature of my teaching approach is to train students in the use of modern software development tools and to have them apply those tools to realistic development tasks. For example, software-development platforms, frameworks, and APIs have become ubiquitous in modern software development. The CS2013 Guidelines have even added Platform-Based Development as a core knowledge area. To address this knowledge area, I been using web application development as the context within which to teach my students software engineering skills. In particular, I have used both Java EE and more recently Ruby on Rails, both of which are industrial-strength platforms with the sizable scale and complexity common in contemporary development platforms. While the use of such platforms helps satisfy the Platform-Based Development knowledge area, they also create excellent contexts to teach students about core software engineering topics, such as software architecture, design, and reuse.

In addition to platforms and frameworks, modern software development also involves considerable use of development environments and infrastructure. Thus, I also make mastery of these technologies a first-class feature of my SE courses. I have trained students in the use of modern integrated development environments (IDEs), such as Eclipse, as well as UNIX-style development environments, which generally involving a variety of tools and editors accessed via the command-line. An essential piece of project infrastructure that I incorporate in my SE courses are version control systems (VCSs). In particular, I have used the VCSs Subversion and more recently Git (and GitHub) as core technologies in my courses. Although some aspects of real-world software engineering are difficult to simulate in the classroom, teaching students these real-world technologies goes a long way toward providing them the preparation to effectively begin their lives as professionals.

2 Educational Psychology and Pedagogical Methods

Helping students to gain the expertise they need to effectively apply SE methods and tools to new problems poses a considerable challenge to educators. To address this challenge, I have looked
to the literature on educational psychology and pedagogy. One key idea from this literature that I have adopted in my teaching is active learning. Studies have consistently shown that passively listening to traditional didactic lectures is not conducive to student learning, and their ability to assimilate knowledge drops off sharply after roughly 20 minutes of continuous lecturing. Active learning engages students through activities and discussions in class, and emphasizes higher-order thinking and often involves collaborative group work. To apply active learning, I break my lectures into roughly 20-minute mini-lectures, interspersed with short (e.g., 5–10 minutes) activities, which often involve problem solving or critical analysis. For example, I frequently use Think-Pair-Share activities in which students first think quietly about a question or problem, then discuss their ideas with a partner to arrive at a solution, and finally, are randomly selected to share their answer with the class. In an era when student distractions and disengagement in class are at an all-time high, I find that these active learning techniques are essential to effective classroom teaching.

Since many of the skills I aim to teach students involve complex technical tasks, I also rely heavily on pedagogical methods for teaching problem solving. In particular, I apply methods based on Cognitive Load Theory, which holds that by minimizing extraneous mental load during instruction, a learner can keep more relevant information in memory, thereby producing better learning outcomes. One such method is the use of worked examples, wherein an expert walks a novice step by step through the process of solving an example problem. In doing so, the expert helps convey his/her mental model to the learner, which studies have shown produces better outcomes than, say, if the learner figures out the problem on his/her own. Since many development tasks are lengthy and complex, one effective way that I provide worked examples is through the creation of demonstration videos in which I perform example tasks. 1 Thus, students can watch and re-watch the videos, and discuss particular parts with me if they have questions. Another method from Cognitive Load Theory that I use heavily is subgoal labeling, which involves grouping the steps of a worked example into meaningful labeled units. By decomposing the process into such units, students are better able to identify structural information and are less distracted by incidental complexities. With the help of these methods, I find that students are better able to learn and apply the complex methods and technologies that are so essential to SE.

3 Course Preferences

I look forward to teaching courses on software engineering and human-computer interaction at all levels. The core subtopics that I most enjoy are software design, and requirements and usability engineering; however, I am comfortable teaching other core subtopics, such as software engineering process, testing, and evolution. As special-topic and elective courses, I am most excited to teach empirical software engineering, program comprehension, recommender systems for software engineering, and agile web development. I also have experience with and am comfortable teaching a number of lower-level courses, including introductory CS1, CS2, and data structures, as well as programming languages and compilers. Finally, as an Assistant Professor, I have been teaching the CS capstone project course for several years, and have learned a great deal about how to work with industry to provide students effective professional development and project experiences.

1Sample video: https://youtu.be/TKRiazcHecQ?list=PL0s90BqgiDzyJfWcuEq0enJ4RcalqDCQC