Session: FA3-2

DEEPER THINKING: ONE APPROACH FOR TEACHING COMPUTER PROGRAMMING TO UNDERGRADUATES IN NUMERICAL METHODS COURSES

Matthew T. Balhoff
Department of Petroleum and Geosystems Engineering
The University of Texas
balhoff@mail.utexas.edu

Kathy J. Schmidt
Director, Faculty Innovation Center
University of Texas at Austin
k.schmidt@mail.utexas.edu

Abstract

Most engineering curricula involve an introductory course on computer programming and skills along with numerical approaches to solving engineering problems. Often students enter these courses frightened and intimidated by computer programming because it represents a deviation from traditional science, math, and engineering classes. To the students, this type of course is intimidating because they may enter the classroom without the necessary foundation (high school preparation can be inadequate and non-standardized) and now they are going to be held accountable for knowing a more logical, step-by-step approach to solving problems. Here, we present a number of unconventional approaches for teaching programming that are intended to get the students to "dig deeper" and to learn how to use programming to think. Many of these methods involve "active" learning (e.g. writing computer programs along with the lecture) and "collaborative" learning (e.g. small group problems during class). Many innovative approaches involve problems and teaching styles that are playful and help maintain interest among the students. For example, the students are given programming projects in which they are required to mimic well-known computer rankings in college football or play a game of "blackjack." We will discuss these methods and ways we are measuring their success.

1. Introduction

Engineering students are usually required to take an introductory course on computer programming and skills along with numerical approaches to solving engineering problems in the first two years of the curriculum. The need for these skills is fundamental in today's world, yet many engineering students are unprepared and frightened by computer programming because it represents content that deviates from previous courses To the students, this type of course is intimidating because they may enter the classroom without the necessary foundation (high school preparation can be inadequate and non-standardized) and now they are going to be held accountable for knowing a more logical, step-by-step approach to solving problems [8] .

Many students, however, simply hope to “get by” with a passing grade and never program again (the co-author, Balhoff, was once one of those students). The results are often disastrous; students often fail the course altogether or worse, they do manage to just “get by” and retain almost none of the computer skills necessary to be a successful engineer. Furthermore, they often struggle in subsequent courses because they lack computational skills [3].

In our efforts we include unconventional teaching methods in introductory programming courses in engineering in order to reach out to students who struggle with programming. The authors believe that given the nature of computer programming, it should not be learned passively (taking notes while the instructor lectures) but actively (thinking and working during the lecture). Each semester the authors have incorporated more proven “active learning” [1], “learning-by-doing” and “team-based” [5] methods to the classroom with great success (grades have improved on more difficult tests). They have also implemented one-on-one private instruction with every student (80-90 students/class) which has resulted in improved confidence and skills of the students. Finally, examples of fun and creative applications outside engineering are introduced which maintain interest of students who have not yet been exposed to many engineering courses. Here, we discuss in detail some of these methods that continue to be developed and improved.

2. Methods and Results

2.1 Fun Applications

Most numerical methods/programming courses are offered to 2nd year students, well before they take most of their engineering courses. Students have not learned enough “real-world” engineering applications to offer many projects and homework problems. However, students often struggle to maintain interest for computer programs and numerical methods that are not associated with a real application. It is often less than engaging to hear about a computer program rather than to experience it. One alternative is to present real-world applications that everyone (not just engineers) is familiar, many of which are fun.

One of the first concepts taught in most programming courses involves loops (“for” and “while”) and branching (“if” statements). Many students struggle with the correct use of these programming tools when first presented. Card games (poker, blackjack, solitaire, etc.) are popular amongst students as computer games; they are excellent examples of applied computer programming. For example, blackjack (“21”) requires multiple “while” loops and decisions by the player (“if” statements). A group project is given early in the semester for students to write and play their own blackjack program. Feedback on the project has been very positive; students claim to understand how loops are used after completing the project. If assigned later in the semester the project can be made more challenging by requiring a Graphical User Interface (GUI) or offering a more difficult card game (e.g. poker).

The solution to systems of linear equations is an important topic learned in numerical methods classes. They are applicable in many science and engineering problems (e.g. circuits in electrical engineering, interconnected reactors in chemical engineering, and reservoir simulation in petroleum engineering), but 2nd year students are usually unfamiliar with these applications. An alternative real-world example involves rankings for college football, a popular pastime for students at the University of Texas and many schools across the country. Teams are interconnected by the games they play, similar to nodes being interconnected in an electrical circuit.

Computer rankings are used in college football to help determine a national champion. These computer rankings are often viewed as “black boxes” and most casual fans have little understanding of how they work. The Colley Matrix [2] is one computer ranking system which relates a teams ranking to its schedule and record by mathematical formulas. The approach results in a system of over one hundred simultaneous equations for each team and solution to those equations results in the ratings for every team in college football. Students are asked to write a computer program to read in data of college football, write loops to set up the system of equations, solve the equations using methods taught in class, and then print out the final rankings. The students verify the results by comparing to various media outlets, such as ESPN.com. The project has received great interest from the students.

2.2 One-on-One Demonstration

It is well known that students learn better in small class sizes and even better in a one-on-one environment. Unfortunately, engineering courses are usually quite large (50-150 students per class). In these large classes, students are less likely to ask questions and as a result may fall behind. Because of the nature of computer programming, it is especially difficult for a student to “catch up” once he/she falls behind. Students unwilling to seek help may quickly become frustrated and give up in the class altogether.

An unconventional instruction and testing approach is utilized here which involves one-on-one demonstration meetings with the instructor. 20 minute demonstration meetings are set up for every student (one meeting per semester) to demonstrate their computer skills. The meeting serves several purposes:

1. Allows the instructor to get to know his students on a more personal basis and for them to get to know him.

2. Provides an additional (perhaps more accurate) testing method. Students write and debug programs on a computer in the instructor’s presence instead of handwritten problems on a timed test. We don’t all think at the same speed and this approach allows for more optimal thinking.

3. Offers immediate feedback. Students are corrected as they write the programs and once they finish. Students are explained concepts that they have been missing all semester.

4. Presents an opportunity for students to ask specific questions, when they may not have stopped by office hours otherwise.

The one-on-one demonstration has proved to be an extremely effective testing and instructional method. Students are clearly seen to “get it” in these meetings and class improvements are visible. Test averages have been found to be noticeably higher for students who have completed their demonstration grade. On course evaluations, students have requested more demonstration tests and instructor ratings have greatly improved since their implementation.

2.3 Active and Cooperative Learning

Active and cooperative learning have become popular and effective teaching methods in engineering over the last two decade [4,5]. Active learning which refers to students be actively involved in their thinking. For engineers this implies "learning by doing" and in our case we refer to short (usually less than 3 minutes) in-class problems where students are required to answer a question and are then specifically called-on to respond. Cooperative learning where students work together with a shared goal has been well documented [7]. While active and cooperative learning has been successful in a variety of courses and curricula, they are particularly effective in computer programming classes. Students need the opportunity to learn in context and by making a computer programming class come alive, students are more likely to be engaged.

Programming can be frustrating because small errors can cause the program to fail completely. For students, in particular, these small errors can be very difficult to find and debug, and the frustration can lead to their giving up and quitting a problem altogether. One solution that can help them become independent programmers is to present many small, easy to manage, problems during class in which the students can follow on their laptops. Many of these debugging problems may already be worked out in the notes; students just need to retype them. Their confidence in getting these small problems to work is invaluable. Other problems they must tackle on their own and may not be worked out during lecture, but will require the students to do additional thinking (but still relatively simple). By working in small groups (cooperative learning) along with the course teaching assistants who are present in the classroom, the students are learning helpful methods to debug errors in real time. These in-class problems can be a mixture of short (30 second) questions, ~2-3 minute problems, or 15-20 minute activities.

We have found that by calling on students randomly (by name) after the exercise, they are motivated to work diligently to the find the answer. While some students complain they are "put on the spot", the class feedback has been mostly positive. Students are becoming more likely to attend class, remain attentive, and work on in-class problems if they know they may be called on. Being accountable for your learning in real time appears worth the effort. This approach works best if students are given sufficient time to think, however, and to work in a group or with a partner, and if they do not feel pressured.

It has proven to be essential that the instructor knows and recognizes the students (eye contact can be made even before they are called on). A classroom community where the instructor and students have interactions is needed to facilitate this type of active, interactive teaching. Furthermore, the one-on-one demonstration sessions are integral to the learning process for during these session, the instructor works with the students personally and as a consequence the students feel more comfortable in dialog.

3. Conclusions

Courses on computer programming can often be intimidating for engineering students because it is a deviation from the types of math and science problems they are accustomed to. Students become even more discouraged when they are unable to see the applicability in the subject matter and get stuck on small bugs in their programs. If they lack debugging skills, they often give up. Here, we present several unconventional teaching approaches to alleviate students' anxieties and to maintaining interest in programming. While the approaches developed are specifically for engineering courses on numerical methods and programming, most of them can be extended to other courses as well.

An alternative to presenting engineering applications to students (who have little engineering education as 2nd year students), is to present fun computer problems that relate to students' hobbies and social life. Two examples are presented here that involve card games and football rankings that combine programming and numerical methods. The applications have excited students and enhanced learning.

Individual instruction is well-accepted to be an ideal learning environment, but considered impossible when the student to teacher ratio is high (well above 50 in many cases). We propose one short (15-20 minute) one-on-one testing and instruction sessions with each student during the semester. The demonstration sessions accomplish several goals: getting to know students personally, individual instruction, and more accurate testing. Although the meetings can be accomplished in under 2 hours/week for most classes, an alternative would involve TAs organizing them instead. Students have performed better on tests, requested more one-on-one meetings, and delivered higher course evaluations as a result of these meetings.

Active and cooperative learning techniques are also proposed that involve programming on laptops during class. The approach is effective because students immediately apply concepts taught in class and can receive immediate feedback (with the help of TAs and fellow students). Moreover, confidence is improved as a result of successful, short programs written and run. These techniques work best when students are called on (randomly, by name) following short questions or problems.

References

[1] Bonwell, C., and Eison, J. (1991). Active Learning: Creating Excitement in the Classroom. *ERIC Digest*, 1991091

[2] Colley, WN: unpublished (available at http://www. colleyrankings.com /), accessed December 14, 2009.

[3] Collura, M. and Daniels, S. (2008). "How Accurate is Student Self-Assessment of Computer Skills?" *ASEE Annual Conference and Exposition, Conference Proceedings*, (2008), June 22, 2008 - June 24, 2008

[4] Felder, R. (1988). "Learning and Teaching Styles in Engineering Education," *Engineering. Education*, 78(7), 674-681.

[5] Felder, R. and Brent, R. (2002). "Effective Strategies for Cooperative Learning,",*Journal of Cooperation and Collaboration in College Teaching*, 10(2), 69-75.

[6] Golanvari, M. and Garlikov, R. *ASEE Annual Conference and Exposition, Conference Proceedings*, 2008, June 22, 2008 – June 24, 2008.
[7] Johnson, D. W., & Johnson, R. T. (1995). *Teaching students to be peacemakers* (3rd ed.). Edina, MN: Interaction Book Company.
[8] Steyn, T. and Carr, A. (2008*).* "Skill Development Using Logo – Experiences with First Year Engineering Students on an Extended Study Program," *ASEE Annual Conference and Exposition*, *Conference Proceedings*,(2008), *2008 ASEE Annual Conference and Exposition*, June 22, 2008 - June 24.