STATEMENT OF TEACHING PHILOSOPHY

AHMED TAMRAWI

As psychologists say *thinking* is an uncomfortable process, it takes effort. If I tell you to memorize the random number 9102, you may say probably in two days I will forget it, but if I relate this number to something you already know, e.g., If I tell you that the number 9102 is the reverse of 2019 (last year) when COVID-19 pandemic started, I am confident that when I get to meet everyone who reads this will recall the random number and answer promptly. The conversion from 9102 to 2019 which is the fact that we know is a way to *force thinking*. I believe that thinking and learning correspond to the process of building chunks of information and storing further connected information in a long-term memory. Our lives are full of patterns and our thinking and memory classify new situations into previously learned patterns.

The memorable lectures and talks would be the ones that build on these chunks of information that we already thought of or previously learned. Those are the ones that grabbed our attention either through a funny speaker, visually appealing graphics, or self-explanatory stories. The main guiding principle of my teaching philosophy can be summarized with: *make each lecture a memorable, informative, and exploratory time for students.*

All of us watch Youtube videos and most of the time we skip the ads after the 5 seconds grace period, but there are several times when we watch the full ad. This is the power of *attraction* that I look to invest in my teaching to make students curious to see what next, force them silently to think, and make their learning experience unforgettable.

Teaching Experience

I held couple of positions as a teaching assistant in Yarmouk university, Iowa State University and as an assistant professor at Yarmouk University and Birzeit University. As an assistant professor I was involved in teaching *Software Construction*, *Numerical Analysis for Engineers*, *Operating Systems Design* and *Introduction to Programming* courses.

The *Software Construction*[1] course is a graduate-level course with about 20 students. The primary goal of this course is to help students to create higher-quality software that is robust, flexible, extensible, scalable, and maintainable. The course mainly focuses on the best practices at each stage of the software development process that will lead to constructing a higher-quality software. As an entry-level graduate course, we designed the course to benefit students on both the technical and theoretical sides. This was achieved through: (1) visually appealing lecture slides with in-class activities focusing on trying out new technologies and tools and experimenting with real-word software repositories, (2) a set of open-ended synthesis and conceptual assignment questions that require thorough research, observations, and working software implementation, (3) research paper writing to enhance professional writing skills, and (4) a final programming project with emphasis on applying the concepts studied through the course to a well-established open-source project with many contributors and various issues open to fix. The open-source contribution would enable student interaction with actual developers and will force them to obey project's rules and style.

[1]`https://atamrawi.github.io/teaching/swen6301`

The *Numerical Analysis Methods for Engineers*[2] course is a 300-level course with about 100 students. The biggest challenge in this course was the diversity of the students' background. I worked with my colleagues on evolving the course materials to make it both accessible and engaging for all students. For example, we added a section to every topic or a group of topics to discuss actual applications of the studied numerical method(s) in different fields. We also added a new course outcome that emphasizes on the applications of numerical analysis techniques in real-world setup and added exam question(s) to test that outcome. Students were really challenged and interested especially that the problems spawn different prerequisite courses. Another example is that we came up with visually appealing graphics to use in our course presentations to grab students' attention and simplify the pure mathematical concepts. To that end, we added proper animations to visually comprehend the different mathematical algorithms especially the ones that require iterative solutions.

The *Operating Systems Design*[3] course is a 400-level course with about 50 students. Back from my days, I always remember the Operating System Design course to be boring because of the dense theoretical information. So, I have decided to revolutionize the course and build it from ground up. From my 5-year research experience with Linux kernel code, I decided to interleave between the theoretical information given in the course and that practical aspects of these information from the Linux kernel. To this end, I authored a set of slides for each chapter of the selected textbook. 20% of the slides contents were only text and the rest were images supported with proper animation intertwined with funny jokes and situations that stem for local culture traditions or metaphors that were applicable to the studied topic. I tried to leverage local characters and use them to tell a story of an operating system concept. For example, the story of two cartoon characters who got into a big fight around printers due to a deadlock. The students were very interested to see how the story evolves. For every topic, I relate the topic to my industry experience and show the students the algorithmic reasoning behind every concept. Moreover, I related the studied concepts to the actual code from a live Linux kernel repository. The course was accompanied with the 3-hour lab every week. I have developed a new material for the lab focused on learning new operating system concepts as well as software design, development, and testing. I taught students how to use Linux-based operating systems, how to create build files for a project, how to use software repositories (e.g., github), and how to write proper test cases for a project. We also discussed various algorithm design strategies to choose the best strategy to develop an efficient solution addressing the assigned requirements.

The *Introduction to programming*[4] course is a 100-level course with about 120 students. This course was one of the toughest and most challenging courses to teach. Not because it is difficult but because students have different backgrounds and no coding skills at all. I started to force the students to think outside the box on how we can teach a kid to do something. I also compared a computer to a simple calculator where it knows how to do simple operations and I asked them how we can use these simple operations to perform averaging operations for 50 grades, how to do multiplication through addition and how to do division through subtraction. Then, we moved up to C++ syntax which is used as the programming language to teach the course. I always prepared couple of programs to write in the class to teach multiple concepts. In class, I open the preferred IDE which is used in the Lab as well and start live coding. We first discuss the problem and how we can achieve a solution and eventually motivate the need for some language constructs. Next, we start writing the program interactively, by posing questions to the students and intentionally making logical mistakes to catch their attention. Many times, I stop and intentionally ask them oh should

[2]`https://atamrawi.github.io/teaching/cpe310`
[3]`https://atamrawi.github.io/teaching/cpe460`
[4]`https://atamrawi.github.io/teaching/cpe150`

we do this or that, is this a less than or greater than. Is this a double array with 7 elements or less than that. After we finish the first draft, we iterate through different approaches to solve the same problem. If there is an erroneous output, we use the Debugging feature which comes at great advantage especially in teaching arrays, pointers and functions calls. The course was accompanied with a 3-hour lab per week. In the lab, I have made a set of tasks oriented around toy-size applications that mimic real-world applications such as coffee shop, salary computation, wind forecasting models, etc. The goal was to emphasize on software design, development, and testing. Course theoretical and practical exams were aimed to asses two outcomes: program comprehension and debugging and programming skills. The first outcome was assessed via couple of output related problems, logical errors fixing and describing a mathematical expression between variables that the program induce. The second outcome was about writing complete programs slightly different from the program taught in the lecture or lab tasks.

Teaching Methodology

I believe that a central goal of teaching a subject is to *nurture the ability of students to inquire and learn the subject themselves.* Achieving this goal is particularly important especially in computer science and engineering fields as both fields are most rapidly changing. Many programming languages and software techniques that we are using today did not exist twenty years ago and most likely will be obsolete or superseded twenty years from now. Nevertheless, the ability to learn can serve a lifetime. From my personal experience, I learned all of Java, C#, and C/C++ programming skills and full stack web development by myself from online tutorials and empirical projects. I believe the success of a career in computer science and engineering largely depends on one's ability to keep up with the rapid innovations happening everyday in the field.

In a *lecture setting*, I really like interleaving theoretical material with live tool demonstrations, slides and animation, and hands-on exercises that students and I collectively solve in class. I use this style in my guest lectures and tutorials in the college of engineering at Iowa State University, Yarmouk University, as well as one tutorial on program verification and analysis tools that I gave at Amazon campus in Seattle while I was an intern. I found this to be a great way to build a feedback loop between theory and practice and keep the audience motivated and engaged. Many times, I refer students to my work at EnSoft as well as being a software engineer at Amazon. I was telling them many stories that were for sure engaging and opening up their curiosity to see how things get resolved in real-world and how the learned concepts and theories are applied in real-world setting.

One thing that I teach my students is that: *never think that there is a stupid question, these curious questions may lead you to winning the Nobel prize or inventing a new thing.* This has opened up the students to a lot of curious questions to the extent that I needed to note them down to research answers. I was happy that I was able to pull out these questions from students who are still learning the topic.

From my past experience, interactive lecture components such as open-ended problem sets and large projects are better than traditional lectures and quizzes in nurturing the ability of students to inquire and learn the subject themselves. I am a strong proponent of using large open-ended team projects for teaching computer science and engineering subjects. There are at least three major benefits. First, working on a large project encourages active learning. Secondly, there is always a gap between the theory and the practice. There is no better way to close this gap than actually to implement the learned theory. Lastly, doing projects is a more attractive way for students to learn. It gives the students the satisfying feeling that they invented and created something on their own. To ensure the success of a large project assignment, the project should be broken into many step-by-step incremental milestones so that a steady learning curve can be achieved and the course staff can monitor the progress of each team. More importantly, assistance and guidance from the course staff should be accessible for every student along the way so that students do not stray too far during the early stages of the project.

In a *practical lab setting*, I usually attend the whole 3-hour lab engaging students into discussion and helping out attending teaching assistants. Sometimes, students feel shy or looking for ideas, I simply grab a chair and help the students. This was self-rewarding and relaxing for me and the students. Top-notch students were challenged with bonus questions that require more research and effort. In the lab, I have an open-resources policy where students can collaborate to an extent and can use available online resources to solve the given problem.

In an *exam setting*, I noticed that visually appealing exams are a factor for student happiness and better scores. Also, injecting funny questions after a hard question, make the student step aside from the overwhelming technical concentration to think outside the box and free their minds.

I engage with the students in an *after-exam* session after grading the exam and looking at the ABET analysis results of the exam. In the grading process, I note down where each student has problems or should have solved the problem based on his past performance. After the exam, I compare the current results with previous semesters to gauge my teaching and what is missing in each class. The ABET experience gave me a great way to gauge my teaching. Once the exam is graded, I call each student the following day and ask him/her to meet me at the office. Going through the exam point by point with the student and reading through my grading notes, I start asking the student why did he/she answer this, what changes his/her mind from the correct answer. One time, I realized that the students found one of the question statements to be ambiguous, and on rare occasions, I noticed that students has some wrong interpretations of some concepts. Overall, it was an exciting learning experience.

Teaching Interests

Definitely, I would enjoy teaching both undergraduate and graduate courses on programming languages, compilers, software security and analysis, and software engineering. In addition, I am interested in contributing to introductory programming or algorithms and data structures courses. In particular, I would be interested in bringing elements of program analysis and security into those courses. I would also be happy to teach undergraduate classes outside my immediate areas of interest.

I would enjoy teaching an undergraduate course on *The Mechanics of Programming*. In this course, students will be introduced to the details of program structure and the mechanics of execution and translation processes (e.g., some essentials programming languages and compilers) as well as supportive operating system features (e.g., some essentials of operating system design and concepts). Moreover, this course will shed the light on some security and performance issues in program design (i.e., software analysis and security).

At the graduate level, I would like to teach classes on software analysis and security, and software testing and maintenance. I believe that my extensive research experience in several areas of program analysis and security puts me in a position to design a course that covers a wide range of topics. I have already designed a course based on the book "secure programming with static analysis.". Moreover, I want to offer a crucial course to both graduate and undergraduate students called "Bridging the Academic-Industry Gap" where I will focus on putting all puzzle pieces together from building a resume, preparing for a technical interview, recap data structure concepts, software design, software testing, and software engineering. All of this wrapped up in the final year to make sure the graduating students are ready to be in the position they are dreaming of.

Advising Approach

From my assistant professor time to when I was a PhD student, I was responsible for advising many students either on the graduate or undergraduate levels. Both with different needs. I have always enjoyed working with students in advising capacity. It is like you juice out everything that you know and give them the shortest way to achieve the best. I am an engaging person, so I talk to students on their future career and guide them to the proper way with the help of other professors. As a graduate student, I supervised two undergraduate research projects and two master theses. This humble experience made me realize the importance of finding a good balance between letting students pursue their own ideas and giving them the guidance they need to succeed, in the form of weekly discussions, intermediate deadlines, or reading lists. I also realized that every student is different, and a single style of guidance does not fit all.