Eric Roberts
MLA 321

Handout #5
January 26, 2016

Project Ideas

When you learn about programming, you don't get very far by reading books or listening to lectures. You have to sit down at the computer and write a few programs on your own. Recognizing the intellectual dimension of computer science takes a similar investment of time and energy. You have to wrestle with the ideas. For this reason, I expect that most of you will get even more out of the work you do for the class project than you do from the material I present in class.

The project deliverables are as follows:

- *A 15-20 page paper that describes your topic.* The paper should be a standalone overview of what you would like people to know about your topic, written for an audience that has no special expertise in computer science beyond what one would have by going through the MLA 321 course reader. As in my chapters, illustrations can be extremely helpful in making difficult concepts clear. I won't be in any sense a stickler with respect to form, but your paper should include a list of the references you use.
- *A 15-minute oral presentation to the class.* The goal of the presentation is to give your classmates an overview of your topic area, illustrate the topic with some simple examples, and offer a sense of why that topic is intellectually exciting. (In a way, my hope is to simulate a conference presentation, where the goal of the talk is in part to get the audience excited enough to read the paper.) You are free to use projection technology to illustrate your presentation, but it is important to avoid becoming so wrapped up in the technology that the main points are lost. As shown on the syllabus, the presentations will take place during the last class meeting on March 8.

Suggested topics:

1. *The Antikythera mechanism.* In 1900, a team of divers salvaging artifacts from a shipwreck found fragments of an ancient computing device that now bears the name of the Greek island nearest the site. Over the century since its discovery, archaeologists discovered that the Antikythera device (which appears to date from the 2nd century BCE) was capable of extremely advanced astronomical calculations, including, for example, determining the phase of the moon, tracking the motions of the planets known at the time, and predicting lunar and solar eclipses. The operation of the device itself—and the detective work that went into figuring out how it worked from the surviving fragments—are one of the most fascinating stories in the history of computer science.

   A report on the Antikythera mechanism should include:

   - A discussion of how scientists now believe the Antikythera mechanism worked.
   - A review of the scientific detective work necessary to work those details out.

- A summary of the unresolved questions about the device, including who might have built it and where it was produced.

2. *The invention of computers: Who gets the credit?* For most of the second half of the 20th century, credit for creating the first electronic computer in the United States was generally assigned to J. Presper Eckert and John Mauchly, who developed the ENIAC computer at the Moore School of the University of Pennsylvania in 1945. For their work, Eckert and Mauchly applied for and received a patent on the ENIAC design.

   The question of who actually invented these concepts, however, remains a subject of controversy. From 1937 to 1942, a research team at Ohio State University consisting of John Atanasoff and Clifford Berry developed a computer—usually called the ABC or Atanasoff-Berry Computer—that had many modern features, including the use of binary arithmetic. John Mauchly corresponded with Atanasoff beginning in 1940 and, in 1941, spent several days at the Iowa State campus learning about the machine.

   In 1973, after a protracted legal battle, a federal judge invalidated the ENIAC patent and gave credit to Atanasoff for the invention of the digital computer. The issue, however, remains controversial, with many arguments as to how much the Eckert-Mauchly team actually took from the earlier design.

   A project on this topic should include the following topics:

   - A discussion of the structure and operation of the ABC and the ENIAC, at a level that allows the audience at the presentation to understand how they work.
   - An analysis of the similarities and differences between the two systems.
   - A synopsis of the legal history.
   - A discussion of other areas in which credit is disputed for early progress in computation. In this discussion, you might, for example, consider whether the stored-programming concept is indeed original to John von Neumann and the fact that the designers of computer hardware, such as Eckert and Mauchly, received widespread attention while the people who programmed those systems, almost exclusively women, were largely ignored.

3. *Proofs of program correctness.* In many areas in computing, practice has been improved substantially by exploiting theoretical results. One area in which the success of this marriage of theory and practice has been more mixed is in the field of program semantics, which seeks to capture the meaning of a program in mathematical terms and then to prove that the program implements its formal specification. In a fascinating 1979 article entitled "Social Processes and Proofs of Theorems and Programs," U.S. computer scientists Richard DeMillo, Richard Lipton, and Alan J. Perlis argued that the idea of proving programs correct is fundamentally misguided, generating a firestorm of controversy among computer scientists on both sides of the Atlantic. That paper and its myriad responses in the letters pages of *Communications of the ACM* over the following year (all of which was reprised a decade later after a follow-on article by James Fetzer appeared in the same journal) define what I think is one of the most interesting philosophical debates in the history of computing.

4. *The history of object-oriented programming.* Many of the fundamental ideas of object-oriented programming were developed by the Norwegian team of Ole-Johan Dahl and Kristen Nygaard in the 1960s and are reflected in the design of SIMULA-67 and, to a lesser extent, the even earlier SIMULA I language. What did those ideas look like in the beginning and how have they evolved? Why did it take so long for those ideas to catch on in the mainstream? To what extent are these ideas still controversial?

5. *The magic of CGI.* Recent movies such as *Star Wars: The Force Awakens, Inside Out, Frozen, Toy Story 3,* and *Avatar* have pushed the capabilities of computer graphics to exciting new levels. A lot of the fundamental technology was developed here at Stanford, to the point that three of our faculty members (Pat Hanrahan, Marc Levoy, and Ron Fedkiw) have all received Academy Awards for their technical work. A project in this area could examine the technical issues involved in modern CGI techniques, the effect of those technologies on filmmaking, and likely directions for the evolution of the field.

6. *Data compression.* Even with the enormous amounts of memory available on modern computers, it would be prohibitively expensive to represent movies by storing the color of every single pixel in every frame of the movie. If, for example, you tried to store this data without some form of compression, a two-hour, high-definition movie would require more than a terabyte of data (one trillion bytes). That volume of data requires a high-capacity hard drive and would be completely inappropriate for streaming. Fortunately, using compressed data formats like MPEG-4 can reduce the storage requirements by two orders or magnitude or more.

   A report on this topic should include at least the following:

   - The difference between "lossy" and "lossless" compression.
   - An overview of at least one classical technique for compressing data, such as Huffman or run-length encoding.
   - A survey of techniques used to compress video data along with the substantially different strategies used to compress audio.

7. *Machine learning.* On March 1, I will talk about the philosophical foundations of Artificial Intelligence (AI), but I will leave the technical content of AI largely untouched. One of the most important techniques that has led to recent progress in AI is ***machine learning***, which is the science of simulating learning processes on a computer. (The Turing article that is part of the assigned reading for that class looks at the question of machine learning in the abstract, but the field has come a long way since then.)

   A project on machine learning should include the following topics:

   - An overview of the history of machine learning, focusing on recent breakthroughs.

- A discussion (with examples) of the differences between *supervised learning* and *unsupervised learning*.
- A survey of the applications of machine learning today.

8. *Self-driving cars.* A decade ago, the Defense Advanced Research Projects Agency (DARPA) organized a competition to see who could build a self-driving car capable of navigating a 142-mile course in the Mojave Desert. No teams completed the course in the first year, but a team from Stanford won the challenge in 2005 with a car named Stanley. Since then, may companies—including Google, Mercedez-Benz, Nissan, Tesla, and Toyota, to name a few—have joined the quest to create practical self-driving cars.

   A report on this topic should cover the following topics:

   - The history of self-driving cars, including the DARPA Grand Challenge race.
   - An overview of the technologies used in self-driving cars today.
   - A discussion of the legal issues involved in putting autonomous cars on the road.
   - A survey of the problems that must still be solved before such cars are practical.

9. *The open-source movement.* Over the last two decades, there has been growing enthusiasm for the open-source movement, which seeks to make software freely available rather than commercial. Individual success stories, such as Linux, Mozilla, and Apache, have intensified this trend. A project on this topic might cover the following issues:

   - The philosophy of the open-source movement and an analysis of how that philosophy has evolved since the publication of *The GNU Manifesto* in 1985.
   - The scale of the open-source movement today.
   - An analysis of the underlying economics, focusing in particular on why open-source appears to have been successful at all, when essentially all economic analyses predicted its failure.
   - An assessment of its impact on the software industry.
   - Some investigation of the sociology of the open-source movement and the nature of the people it attracts. It was sobering, for example, to read a paper at the Grace Hopper Celebration of Women in Computing several years ago that suggested that less than one percent of the participants in the open-source community are female—a percentage far lower than the already scandalously low percentage in the industry.

If none of these topics excites your group, feel free to think of something else.