Eric Roberts Handout #16
CS 54N October 24, 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 25-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. You are encouraged to use projection technology to illustrate your presentation—presumably including parts of the web site described below—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 7:00 to 10:00P.M. final exam slot on Thursday, December 15.
- *A web site that gives a more detailed presentation of the topic.* Your web site should include a top-level general introduction along with links to more detailed information, some of which will be part of your site and some of which will be resources elsewhere on the web.

There are also the following milestones:

Wednesday, November 2	*Groups and topics due.* Your first milestone is to assemble a group of three and collectively agree on a topic. In the past, most groups have chosen one of the topics described in this handout. You may, however, come up with a topic of your own, but only if you convince me that your topic offers the same level of challenge and embodies a similar level of intellectual excitement.
Wednesday, November 16	*Annotated bibliographies due.* The first step toward researching your topic is to look up resources on the topic from a variety of sources: books, journals, the web, and faculty members in the department. Make a guess at the resources you'll be using, write it down, and mail it to me.

Suggested topics:

1. *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.

2. *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.

3. *Excursions in a two-dimensional world.* In 1884, Edwin A. Abbott published *Flatland: A Romance in Many Dimensions*—a fantasy about higher dimensional spaces (along with a fair bit of social satire) that has never yet gone out of print. Over the years, other authors have taken up the same thing, as in *Sphereland: A Continuing Speculation On An Expanding Universe* written by Dionys Burger in 1965 and the more recent *Flatterland: Like Flatland Only More So* published in 2001 by Ian Stewart. While these books are primarily concerned with mathematics, there is one adaptation—*The Planiverse: Computer Contact With a Two-Dimensional World* written by A. K. Dewdney in 1984—that has particular relevance for computer scientists. In *The Planiverse,* Dewdney describes how it is possible to build logical circuits and computer hardware in a two-dimensional world.

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

- An overview of the story.
- A discussion of engineering issues in the Planiverse and an explanation of how things work there, particularly including the logic circuits.
- At least one original engineering design for the Planiverse beyond what Dewdney describes.

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. *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.

8. *Deep learning.* In the week after Thanksgiving, I will talk about the philosophical foundations of Artificial Intelligence (AI), but will leave the technical content of AI largely untouched. One of the most important techniques that has led to recent progress in AI is ***deep learning,*** an exciting area of research that combines machine learning, neural networks, and big data—each of which is a hot topic in its own right.

   A project on deep learning should include the following topics:

   - A history of neural networks, which seek to use biological systems of neurons as a model for implementing intelligent behavior.
   - An analysis of how the ongoing increases of computational power have made neural network models more effective in recent years.
   - A discussion of what the adjective "deep" signifies in today's "deep learning" world.
   - Examples of how deep learning has been applied to interesting, real-world problems.

If none of these topics excites your group, feel free to think of something else. Here are a few ideas, which clearly need to be fleshed out a bit:

- The evolution of numerical algorithms, both before and after the rise of computing
- Symbolic mathematical manipulation in systems like Mathematica
- Strategies for managing concurrent execution
- Design and development of special-purpose graphics hardware