A History of Computing Course with a Technical Focus

Geoffrey M. Draper, Robert R. Kessler, and Richard F. Riesenfeld
School of Computing, University of Utah
50 S. Central Campus Dr. #3190
Salt Lake City, UT 84112 USA
{draperg, kessler, rfr}@cs.utah.edu

ABSTRACT
Many courses on the history of computing are designed for a general student audience, and as such, include fewer technical details than one might find in a typical CS course. While this approach is appropriate in some contexts, it risks losing the interest of the students who could perhaps benefit most from the subject — namely, future computer scientists. This paper describes a technically-oriented History of Computing course which we taught at the University of Utah in 2008. Like other History of Computing courses, ours included a significant amount of writing and discussion. However, inasmuch as our course was created specifically for CS students, we also incorporated several “hands-on” programming exercises and demonstrations, giving students actual experience with the computing environments of the past. Students and faculty alike have responded enthusiastically to this dual-faceted approach.

Categories and Subject Descriptors
K.2 [Computing Milieux]: History of Computing—hardware, people, software, systems, theory; K.3.2 [Computing Milieux]: Computers and Education—computer science education, curriculum, literacy

General Terms
Human factors, Design, Languages

Keywords
History of computing, Computer science education.

1. INTRODUCTION
It is widely acknowledged [1, 14, 15] that the history of computing is an important, but often neglected, facet of computer science education. Indeed, when one considers the demographics of today’s college students, the need for a course in computing history is clear [16]. Most students born within the past 20+ years have likely never used a computer that did not have a mouse, or that had less than 256 colors, or less than 1 megabyte of memory. If students are not familiar with the history of computer science, including why certain decisions were made and what the consequences were, they may mistakenly think that “it’s always been this way”—or worse yet, that “it has to be this way.” Either one of these beliefs could stifle future innovation in computing, and perhaps doom tomorrow’s technologists to repeat the mistakes of the past.

As a proactive step to reverse this trend, some universities have started to introduce the history of computing into their CS curriculum. Impagliazzo et al. [15] suggest three possible methods for accomplishing this:

- integrate history into existing CS courses
- offer a general education History of Computing course for non-CS students
- offer a specialized History of Computing course for CS students

While the first of these options is tempting, it can be difficult to manage — every course offered by the department would need to be evaluated and adjusted [22]. In the wake of decreasing enrollment in CS programs in many universities, some institutions have adopted the second approach as a way to pique the interest of students who might not have otherwise considered CS as a field of study. In order to make the material accessible to a broad spectrum of students, such courses often sacrifice technical detail in favor of high-level overviews, biographical sketches, and lessons in economics. Granted, a multidisciplinary approach is essential to any History of Computing course. However, we have observed that most CS students are, in fact, eager to learn the details about how old computers worked. Thus, a course which lacks substantial technical content may risk losing the interest of our best students.

With this in mind, we have developed a one-semester course entitled History of Electronic Computing at the University of Utah, targeted for juniors and seniors in CS. As with History of Computing courses at other institutions, a central focus of the course was on the socioeconomic factors surrounding the development of computer technology, as well as the key people and companies that pioneered this work. However, we also purposely incorporated an undercurrent of technical depth into the course — including details about the architecture of old computer hardware as well as in-class demonstrations and homework assignments.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE ’09, March 3–7, 2009, Chattanooga, Tennessee, USA.
Copyright 2009 ACM 978-1-60558-183-5/09/03 ...$5.00.
requiring the use of historic programming environments and platforms.

Admittedly, such an approach is not feasible in every situation. There are times when a less technical format is preferable; for example, if a History of Computing course is being considered for eligibility as a general education requirement for a university. Cortina and McKenna [6] describe a compelling design for a History of Computing course which is accessible to students outside the core CS discipline. Similarly, the course outlined by Giangrandi and Mirolo [12], while geared towards CS students, likewise emphasizes high-level historical themes. For additional examples, Impagliazzo et al. [15] present three actual syllabi from past university-level History of Computing courses.

In this paper, we outline the design and structure of our History of Computing course, with particular emphasis on the course’s technical content and programming assignments. Students’ response to this course’s mixture of history and technology is also discussed. Finally, we cite some freely-available online resources for incorporating demonstrations of historic computing platforms into the classroom, and present our conclusions.

2. COURSE TOPICS

We used Ceruzzi’s *A History of Modern Computing* [5] as the textbook for our course. We felt that it covered the material at a sufficient technical depth for our purposes. However, the book is limited to the years 1945–2001, with an emphasis on commercial systems sold in the United States. To broaden the scope of our course, we also covered some pre-electronic and electromechanical computing devices, as well as a sampling of electronic computers developed in Europe and Asia. We also discussed some of the late-breaking topics in 21st century computing, such as the ubiquity of personal digital assistants (PDAs), smartphones, and portable music players.

An abbreviated outline of course topics is as follows:

- Mechanical and electromechanical computers
- Early electronic computers
- Early commercial computers
- Transistors, core memory, disk storage
- High-level programming languages
- Operating systems
- Advent of computer science and software engineering
- Minicomputers
- Integrated circuits
- Computer science education
- Calculators
- The microprocessor
- Personal computers
- Portable computers
- Workstations
- Networking, Internet, World Wide Web
- Open-source movement
- Handheld computing devices

For the in-class demonstrations, our original intent was to bring in actual hardware. However, in the majority of cases, this was not practical, especially for older machines whose few existing models are stored in museums. Thus, we relied heavily on software-based emulators to demonstrate the workings of old computers [26]. We include a list of some of these emulators in Section 6.

3. STUDENT ASSESSMENT

3.1 Writing

Similar to History of Computing courses at other institutions [6, 13, 18], ours was a writing-intensive course. The students wrote several short response papers on various topics throughout the semester. In addition, we assigned one term paper-length essay in which students could explore in-depth a subject of their choice. Following a suggestion from Davis [7], we divided this term paper into multiple smaller assignments. This had at least two benefits. First, it diminished the opportunity for procrastination, thereby reducing student stress. Second, it allowed us to give the students early feedback on their writing, engendering a better-quality final paper for us to read, and a higher overall score for the students.

3.2 Programming

We believe that programming is an essential component of a computer science education. For this reason, we incorporated two programming assignments into our course, involving the use of old programming languages and environments. These assignments required significant effort from the students, most of whom learned to write code using modern object-oriented languages and GUI-based integrated development environments. Yet, almost without exception, they welcomed the challenge and performed well, often exceeding our expectations.

3.2.1 Fortran

About a month into the semester, the students were assigned the task of writing a program in Fortran. We devoted a single lecture to reviewing the basic syntax of Fortran, and demonstrated a few code samples using the GNU Fortran compiler (gfortran) in Linux.

The assignment was to build a simple auction/bidding program loosely based on *Masterpiece* [4], a public domain game written in BASIC in the early 1980s. Detailed specifications for the program were provided to the students, who had one week to complete the assignment. Most students’ submissions were between 100 and 200 lines of code — certainly not a “long” program by most standards, but sufficient to give the students a taste of the syntax and idiosyncrasies of an important language designed over half a century ago.

3.2.2 Pre-GUI environments

For the final group project, students were given the option of either doing a 15-minute oral report, or implementing a sizable program on an old platform. To our surprise, the
majority of students chose to write a program, even though an oral report would likely have been a less onerous option. The specifications for this assignment were to implement an “electronic storybook” demo for a science fiction story of their own creation. Their program had to use color graphics and animation, and could be written in any programming language. The chief requirement was that it could not be written for any platform that uses a window-based GUI. Although they were welcome to use any number of systems, our students universally gravitated to either MS-DOS (via DOSBox [9]) or the Commodore 64 (emulated via VICE [25]). Little or no class time was devoted to reviewing the programming environments of these platforms; students were expected to seek out and learn from whatever documentation they could find.

This assignment was especially difficult for those teams who selected the Commodore 64 platform, since its programming environment is quite different from those found on today’s systems. In spite of this, we were pleased that most students went to great lengths to learn the subtleties of this platform. For example, many teams taught themselves how to use sprites (a form of hardware-supported animated bitmap), an impressive feat considering that Commodore BASIC lacks a direct API for this, requiring instead the explicit POKEing of values into memory. One team went so far as to implement full-screen high-resolution graphics in assembly language (Figure 1).

This experience suggests to us that, when given a choice between a programming assignment and a non-programming assignment, most CS students choose to program — even when it means plunging into archaic and poorly documented computing environments.

3.2.3 Student Feedback

Student comments regarding the inclusion of programming into a history class were overwhelmingly positive. A sampling of student feedback is given below.

“I gained a much better understanding for what it was like to be a programmer during some of the periods we talked about in class. I think hands-on is always better than book learning.”

“That was a great idea. It allowed us to have a hands-on approach. The interactivity with emulations of computers in history also engaged our minds in different ways that paper writing alone could not achieve.”

“It gave us more appreciation for the tools we have today.”

As additional anecdotal evidence, one student approached the instructor several months after the course had ended, and announced that his experience in this course had helped him to get a job. Apparently, the interviewer was a seasoned technologist who was impressed by the student’s ability to converse intelligently about older computer systems.

4. CLASSROOM MECHANICS

4.1 Lecture Format

The class met three times a week, with each session 50 minutes in length. Despite the relatively large class size (41 students), we strove to maintain a discussion-oriented classroom atmosphere as much as possible. To this end, we intentionally took a minimalist approach to our lecture slides, which consisted mainly of short bullet points. These served primarily to guide the discussion, rather than present the course content in an exhaustive manner. Although we posted our lecture slides to the course web site, students quickly realized that viewing the slides online was a poor substitute for attendance and participation in classroom discussions.

To maintain a classroom environment conducive to discussion, students had to be both physically present and mentally engaged. To this end, we had a “no open laptops” rule during class. This policy met with mild objection from a few students, but overall was respected. As the semester progressed, however, we found that computers may have a place even in a discussion-oriented class, for example, when a student asks a pertinent historical/factual question for which neither the instructor nor other students know the exact answer. In these cases, we found it helpful to have a daily “designated Internet searcher” and let the students rotate through this position.

4.2 Guest Speakers

Guest speakers are an important component of many History of Computing courses [6, 18], including ours. We are fortunate at the University of Utah to have a number of senior faculty with experience using historic machines such as the LGP-30, IBM System/360, and DEC PDP-11. They were more than willing to share their experiences with our students, and often brought in pieces of equipment from old computers.

These guest lectures were enjoyable not only for the students, but the speakers themselves also seemed to appreciate the experience. We found that seasoned technologists often have sentimental feelings for the machines they used early in their careers, and are eager to talk about them. Each of our guest speakers thanked us profusely for the “privilege” of speaking to our class, and offered to return.
Table 1: Student Course Evaluation

<table>
<thead>
<tr>
<th>Question</th>
<th>(this course) SA/A</th>
<th>(all CS) SA/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>The course objectives were clearly stated.</td>
<td>50.0%/46.7%</td>
<td>43.3%/39.2%</td>
</tr>
<tr>
<td>The course objectives were met.</td>
<td>56.7%/40.0%</td>
<td>47.0%/38.6%</td>
</tr>
<tr>
<td>The course content was well organized.</td>
<td>55.2%/37.9%</td>
<td>46.3%/34.7%</td>
</tr>
<tr>
<td>The course materials were helpful in meeting course objectives.</td>
<td>60.0%/36.7%</td>
<td>45.4%/34.2%</td>
</tr>
<tr>
<td>Assignments and exams reflected what was covered in the course.</td>
<td>70.0%/26.7%</td>
<td>50.0%/36.3%</td>
</tr>
<tr>
<td>I learned a great deal in this course.</td>
<td>56.7%/40.0%</td>
<td>41.9%/31.8%</td>
</tr>
<tr>
<td>Overall, this was an effective course.</td>
<td>53.3%/40.0%</td>
<td>49.3%/34.6%</td>
</tr>
</tbody>
</table>

4.3 Daily Test Questions

Inspired by [7] and [19], we decided to experimentally employ a slight variation of the “minute paper” concept in our classroom. The minute paper, first described in [8], is a respected strategy for evaluating students’ understanding of course material in a low-stress manner. However, instead of asking students to submit a brief summary of the main point of the day’s lesson, we asked the students to submit a possible “test question” that could be answered from the material presented that day. We devoted the last 3 to 5 minutes of each class session to this activity, to ensure that students had enough time to write a well thought-out question.

We then took a few minutes after each class to read through each of the submissions for that day, and compiled a list of those questions that we felt corresponded to the main points of the lesson. In this way, we gathered a substantial bank of test questions. On the downside, however, some students did not take this exercise seriously and wrote overly simplistic questions. Most of the time, though, the students’ questions were quite thought-provoking, revealing insights into the material that we had not necessarily considered.

To compile the midterm and final exams, then, we simply selected a subset of the questions from the student-generated test bank, and augmented it with a few questions of our own. For each student-generated question that we used in an exam, we rewarded the student with a token amount of extra credit. In this way, students had some motivation to write high-quality questions.

5. COURSE EVALUATION

Quantitative feedback from students for the course as a whole was quite positive. Table 1 shows students’ responses to the university’s course evaluation questionnaire. The first column indicates the percentage of “Strongly Agree” (SA) or “Agree” (A) responses for this course, and the second column shows the average percentage of SA/A responses for all CS courses taught by the department that semester.

6. RESOURCES

Fortunately, there are many online resources related to the history of computing. What follows is a partial list of those we used in our course. Each of these resources is freely available for academic use.

- The 3D Pascaline simulator [2] by Ioannis Anastasiadis provides a reasonably accurate replica of Blaise Pascal’s mechanical calculator. We used it to perform basic arithmetic as part of an in-class demo early in the semester. The simulator is written in VRML, so it requires a browser plug-in (such as Cosmo Player [10]) to view it correctly.

- Dirk Rijmenants’ well-known Enigma simulator [21] faithfully reproduces the look and feel of a 3-rotor Wehrmacht and Luftwaffe Enigma machine, as used by the German military in World War II. Our students’ first homework assignment was to decode an encrypted message using a simulated Enigma machine, and to submit a brief write-up about their findings.

- The Java-based ENIAC simulator [27], written by Till Zopplke, provides a complete graphical reconstruction of the original ENIAC’s control panel. This simulator is highly interactive, allowing the user to virtually “wire” the machine by plugging cables into various ports via a click and drag interface.

- Spacewar, created by Steve Russell in 1962 at MIT, was one of the first video games ever written. Through the efforts of Barry Silverman, Brian Silverman, and Vadim Gerasimov, a version of Spacewar based on the original assembly code is available via a PDP-1 emulator running in a Java applet [23].

- The IBM Archives website [17] is a veritable treasure trove of information about IBM’s mainframe systems that dominated computing for much of the twentieth century. We found this site especially useful for finding vintage publicity photographs and pricing information of these systems.

- The SIMH project [24], led by Bob Supnik, provides emulators for a variety of machines manufactured by DEC and others. We used SIMH in the classroom to demonstrate an interactive session with a simulated PDP-11 running the RSTS operating system.

- The Altair32 simulator [20] by Richard Cini features a highly accurate representation of the Altair 8800’s front panel, complete with toggle switches and LEDs. To kick off our discussion of microcomputers, we used this program to demonstrate the operation of an Altair 8800. Educators wishing to use Altair32 should be advised, however, that this program runs in a rather small window; we had to decrease our screen resolution in order to make the details of the GUI visible on the projection display.

- Finally, there are many free emulators for the 8-bit computer platforms of the 1970s and 1980s. Examples include AppleWin [3], VICE [25], and EightyOne [11]. These programs emulate the Apple II, Commodore, and Sinclair ZX81 machines, respectively.

7. CONCLUSIONS

In this paper, we describe a History of Computing course offered by the University of Utah in 2008. Our course differs from comparable course offerings in that we incorporated a
strong component of technical content into the course, including in-class demonstrations of old computing environments and programming exercises. However, in recognition of the fact that history is a multifaceted subject, we also emphasized the stories behind the people and corporations that shaped modern technology, in the context of the socio-economic climates of their times. A course on the History of Computing is neither wholly computer science nor humanities, and we strove to maintain a healthy balance in this course.

In general, students responded favorably to the course’s mixture of both written work and programming exercises. We hope that the ideas, resources, and experiences described in this paper will be useful to other educators wishing to include a technically-oriented History of Computing course in their CS curriculum. Those desiring specific information about the course can contact the authors directly or visit the class web site at:
http://www.cs.utah.edu/classes/cs4960-02-draperg/

8. ACKNOWLEDGMENTS
This work was supported by the University of Utah’s Graduate School and School of Computing, under the auspices of the University Teaching Assistant (UTA) program.

9. REFERENCES