An Introduction to Computer Science for Non-majors Using Principles of Computation

Thomas J. Cortina
Computer Science Department
Carnegie Mellon University
tcortina@cs.cmu.edu

ABSTRACT
In this paper, the design and implementation of a novel introductory computer science course for non-majors is presented. This course focuses on the major contributions in computer science from the perspective of the process of computation. This course differs from most introductory courses in computer science in that it does not include programming using a computer programming language. Students focus on algorithms and the principle of computational thinking, and use a flowchart simulator to experiment with various short algorithms and build simple computer games without dealing with programming language syntax. Steadily increasing enrollments and interest from various departments on campus indicate that this course has become a successful addition to our introductory CS offerings.

Categories and Subject Descriptors
K.3.2 [**Computing Milieux**]: Computers and Education – *computer science education, curriculum, literacy, self-assessment.*

General Terms
Algorithms, Theory, Human Factors.

Keywords
Computational thinking, computer science education, non-majors, curriculum.

1. INTRODUCTION
In recent years, a number of new courses have been developed to deal with the declining interest in computer science for non-majors nationwide. [8,10,12] These courses use programming with objects or the web as the vehicle to introduce computer science and make the discipline more appealing. These courses still focus on the programming process as the running theme, and not on the broader principles of computer science throughout the course.

Observing my students for over a decade of teaching introductory computer programming, it became clear that for non-technical non-majors (i.e. students in majors that were not purely mathematical, scientific or engineering oriented), the preciseness and detail needed to write computer programs correctly was overwhelming. Additionally, informal surveys with these students indicated that they took the course because they were required to, yet no one expected to use the programming skills in their careers. As educators, we understand that these students are learning problem-solving skills, and that they will use these skills in their careers, but many introductory CS courses focus on the programming language details, especially as the languages we use become more complex and industrial in size.

At Carnegie Mellon University, a new introductory course has been developed that presents the principles of computer science rather than computer programming. Unlike a traditional CS0 course that covers one week of each major course in computer science, this course focuses on the principle of computation, and how computer scientists study computation through the design and analysis of algorithms, correctness and efficiency, the limits of computation, and several unique applications that build on CS ideas (cryptography, artificial intelligence). Students do not program in this course using a traditional programming language. Instead, they use a flowchart simulator to examine how computations work and experiment with various algorithms to study the process of computation. The overall goal is to teach students how to think computationally. [16]

This paper will describe the overall design of the course, the use of guest faculty to provide insight into the future of computation, and a collection of survey results to indicate that this course has become a successful entry in our introductory course offerings in a span of one year.

2. COURSE DESIGN
When considering students who are only going to take one course in computing in their undergraduate curriculum, one may question whether an introduction to computer programming is the best introduction to computer science. After years of teaching introductory programming in a variety of languages, my observations of non-technical non-majors in such a course brought about a realization that these students are taking the course only to satisfy a requirement, and they are not going to program ever again in their lives. These students need to understand the power of computing and the unique problems we face in computer science that can affect their disciplines. Put another way, we would like to show them that computer science

is much more than computer programming, and we only have one semester to do this.

In the fall semester of 2005, I put together a new course for non-technical non-majors titled Principles of Computation. This course is a survey of the major contributions and issues of computer science from the perspective of the process of computation. In this course, computer science is not viewed through a specific programming language nor is it viewed through a specific application area (e.g. multimedia, robotics, etc.). Instead, the focus is on what it means to perform computation and what issues arise as mankind automates this process using computers.

2.1 Course Topics

The course is organized loosely based on the book *Algorithmics* by David Harel (now in its third edition with co-author Yishai Feldman). [11] Since these students are non-technical non-majors, great care has been taken to present the ideas in a more intuitive manner, avoiding the standard rigorous proofs in computer science courses. This is especially important since these students have a level of mathematical maturity that is typically below that of the undergraduate science or engineering major.

The lecture topics include the following:

- The history of computation
    - Early devices, suppression of computational advances in the Dark Ages, Babbage and Hollerith's contributions in the 19th century
    - Advances in computing in the 20th century due to war (WWII, Cold War)
- Expressing computation using algorithms
    - Tracing algorithms expressed in pseudocode
    - Algorithmic constructs: assignments, conditionals, loops, subroutines
    - Flowcharts to visualize computational flow, simulation using Raptor
- Organizing data
    - Arrays vs. lists
    - Stacks, queues, trees and graphs
    - Database terminology
- Expressing algorithms as programs for a computer to execute (automated computation)
    - Programming Language Paradigms (imperative, object-oriented, functional, logical)
    - Compilers and Interpreters
- Computational tricks of the trade
    - Recursion
    - Divide and conquer algorithms (merge sort, towers of Hanoi)
    - Greedy algorithms (Hamming codes, minimal spanning tree)
    - Dynamic programming/memoization (Fibonacci numbers, shortest path)
- Perfecting computation
    - Correctness (use of invariants, principle of induction to prove correctness)
    - Efficiency (order of complexity)
- The limits of computation
    - Intractability
    - Undecidability
    - Universal computation (Turing machines and counter programs)
- Concurrency
    - Multiprocessing (synchronization, maximum speedup, overhead)
    - Pipelining
    - Multitasking (operating systems, deadlock and starvation)
- Applications
    - Public-key cryptography
    - Artificial intelligence (Turing test, game trees)
- The future of computation
    - Guest lectures from faculty on quantum computing, nanotechnology, etc. (varies)

A good deal of this course deals with the preciseness required of computer scientists when expressing computations as algorithms and testing their correctness. Also, students see how computer scientists deal with extremely large numbers when examining efficiency and tractability. Connections are made to modern examples such as Google's search through billions of web pages and modern-day software failures such as Therac-25 and Y2K to illustrate how computer science relates to their own lives.

2.2 Simulating Flowcharts using Raptor

Since students see a number of algorithms in this class, it is important for them to be able to see some of them automated to understand how they work. Students use a flowchart simulation tool called Raptor [2] to build flowcharts of computations and simulate them on a computer. Raptor supports input and output, conditional and loop constructs, subroutines, arrays and graphics. Raptor allows students to easily construct a computation for execution since it has very little syntactic overhead. For example, when a loop is added to a computation, the flowchart is adjusted automatically based on its presence. Students cannot leave out brackets or create a loop with an “empty body” accidentally. Additionally, Raptor provides breakpoints for debugging, controls for the speed of simulation, and a frame to trace variables during simulation to see what a computation is doing.

In our experience, students learn how to use Raptor in one tutorial class in the lab, and then there are only a few questions about how to use Raptor for the rest of the semester. There is hardly any additional class time spent on Raptor, as opposed to other flavors of CS0 for majors that spend more time focusing on some language or programming environment.

Students initially use Raptor to explore several short algorithms and then use Raptor later in the semester to work on a simple game. In one semester, students were given a Raptor flowchart for the popular board game of Connect Four. The game contained a number of subroutines to control various functions such as determining if a player's move is valid, drawing a new checker on the board and checking to see if there is a winner. Students were asked to complete the subroutines based on descriptions of what they had to compute for each.

2.3 Examining Social Aspects

The achievements made in computation today (algorithms, software development and architecture) have led to a dramatic change in the way we lead our lives. It is very important that students explore how computers have created new problems for society to handle. In addition, as educators we all believe that our students should improve their communication skills.

With these goals in mind, this course includes a term paper that addresses how computers have affected mankind in a social context. Previous topics for this paper have included

- MP3 file sharing systems, copyright infringement
- Privacy and security of electronic data, identity theft
- Electronic voting systems, verification and security

Term papers are assigned in three stages, starting with an outline that indicates the major sections and subsections of the paper along with a suggested list of references. Students are required to select at least five references, three of which must come from peer-reviewed or edited print sources. (Students may reference their online equivalents.) A discussion is included in lecture about evaluating references for reliability and accuracy, and there is a discussion about the reliability of information posted on websites like Wikipedia. [4]

Once the outline and references are reviewed, students work on a draft of the paper. This draft is reviewed for grammar and organization. Once the draft paper is completed, students complete their final papers including references in MLA format. This exercise would also be a good exercise for computer science classes that tend to minimize the amount of written or oral communication in favor of programming projects.

By assigning the paper in stages, this minimizes the impact of term paper sites, where students can download fully written term papers for a fee. [13] Since this course is given as a freshman course, it has no college-level writing prerequisite. A term paper like this in an introductory computing course is easier to handle if your college or university has a writing center to assist you with students who have poor writing skills. At our university, there is a lack of writing tutors on campus for students who need significant help with their writing skills, so this burden is shifted to the instructor of the course.

2.4 Student Assessment

Students are evaluated through a number of assessment instruments:

- Written homework and Raptor assignments 15%
- Term paper 10%
- 3 written exams 45% (15% each)
- Final exam 30%

Most of the student's grade comes from exams, minimizing the effect of any collaboration during written homework and Raptor assignments.

3. ADDING ADDITIONAL PERSPECTIVES

The success of a course like Principles of Computation depends on the participation of other members of the faculty. A key component of the course is a set of guest lectures from departmental faculty on the future of computing.

Here is a list of topics presented to students during the last week of the semester, along with references to associated readings. (Not all topics are presented each semester.) These presentations are geared to non-technical students and show them what researchers are working on presently that will shape the future of computing.

- An introduction to quantum computing [7]
- Google: Large-scale distributed search on the web [1]
- Using nanotechnology to build programmable matter [9]
- Using humans as computers to solve problems that are very difficult for computers today [15]
- Computer vision: using robots to play soccer [14]

The selection of topics should take advantage of the expertise in your department. This will add another dimension to the course, and it will allow the department to help spread the word about the world of computer science to students who have a biased perspective due to the overemphasis of learning a programming language as the first introduction to computer science.

4. DATA & FEEDBACK

Response to this new course has been slow at first but has increased dramatically in its second year. A major problem that we had to overcome across our campus was the fact that most departments only had one choice for an introduction to computer science from our department: introductory programming. This meant that advising instructions listed introductory programming as the required CS course even though this new course is now available. Several departments have added or are considering this course as an alternative to the introductory programming course, including business and information systems.

4.1 Enrollment

Due to the historical "tradition" that the first CS course is an introduction to programming, enrollment in this course was initially quite low despite heavy advertisement and meetings with chairs of the targeted academic school on campus. However, as word spread about this course and its overall goals, enrollment began to increase, as shown in table 1.

Table 1. Number of students enrolled Fall 2005 - Fall 2006 (3 offerings)

Semester	Enrollment
Fall 2005	6
Spring 2006	22
Fall 2006	40

Although this course was targeted for students in the school of humanities and social sciences as an alternative to Java programming, the initial push in enrollment came from the business school in the Spring 2006 semester. In the business administration program, students are required to take one course in computer science to expose them to the principles of computing. After hearing about this new course offering, the business school modified its graduation requirements to allow for Java programming or this new computation course, and a number of students opted to take this alternate class.

Continuing this trend, faculty in charge of the information systems major felt a need to give their students a bigger picture of computer science. These students are required to take a number of CS courses, but their adviser reported that most of their students don't understand why they need the CS courses. They feel that the introductory survey offered by this course fills this need well. As a result, enrollment increased again in the Fall 2006 semester due to an influx of information systems majors.

It remains to be seen if additional majors or departments will require their students to take this broader introduction to CS principles in place of, or in addition to, the introductory programming class. Particularly, students from science and engineering majors are required to take a large number of preset courses, and some do have a need to write computer programs, so adding this course will be more difficult.

4.2 Major & Year Distribution

As stated earlier, this course was designed as an alternative to introductory programming for humanities and social science majors. Table 2 shows the distribution of students taking this course by major, indicating that we have indeed seen a majority of our students come from the humanities and social sciences. (Information Systems is housed within the College of Humanities and Social Sciences at our campus.)

Table 2. Distribution of majors by school
Fall 2005-Fall 2006 (65 students total)

School	Percentage
Humanities and Social Sciences	58.5%
Business	26.2%
Engineering	6.2%
Computer Science	3.1%
Science and Mathematics	1.5%
Other (Undeclared, 5-year program, Masters)	4.6%

Although this course was designed to be an introductory course for freshmen (with the hope that some students might get excited about computing and take additional courses later), the distribution of majors by year shows that approximately a third of the students taking this course are not freshmen, as shown in Table 3.

4.3 Grade Distribution

Table 4 shows the grade distribution for the first two semesters of this course, compared the grade distribution of the two semester of introductory programming taught by the same instructor during the same time frame. The table suggests that the increase in enrollment in this new introductory course is not based on the chance to obtain a higher grade. The average grade in both classes was a B, and fewer students earned A's in the computation course, yet the enrollment continues to increase.

Table 3. Distribution of majors by year
Fall 2005-Fall 2006 (65 students total)

Year	Percentage
Freshmen	66.1%
Sophomore	15.4%
Junior	13.8%
Senior	1.5%
Other (5-year program, Masters)	3.1%

Table 4. Distribution of grades for the new introductory computation course compared to the traditional introductory programming course taught by the same instructor
Fall 2005-Spring 2006 (2 offerings)

Grade	Computation (28 students)	Programming (112 students)
A	28.6%	33.9%
B	42.9%	42.0%
C	17.9%	14.3%
D	0%	4.5%
R (fail)	0%	4.5%
W (withdrawal)	10.7%	0.9%

4.4 Some Student Feedback

At the end of each semester, students are given a standard university survey form to rate each course they take. In addition to this survey, a specific anonymous survey was given about this new course. Feedback from this anonymous survey for the Spring 2006 semester (20 respondents) is given below.

Should we continue the use of guest speakers in this class?

Yes	80%	No	15%	Unsure	5%

Should we continue the use of Raptor in this class?

Continue	30%
Continue but use more graphics	55%
Use another language/application	20%
Stay off the computer	5%

Did you take another CS before or during this class?

Yes	20%	No	80%

Now that you have taken this class, are you interested in taking another CS course if you could?

Yes	55%	No	35%	Unsure	10%

Would you recommend this course to your friends?

Yes	85%	No	15%

Describe this course in one sentence to one of your friends (selected responses):

“An alternative/substitute to [introductory programming].”

“You learn a lot more than just how to code.”

“It's a lot better than doing real programming.”

“It's a survey of computer science that emphasizes the concepts and ideas – how computer scientists think about problems.”

“It's programming but you don't just type word[s] for hours.”

“The fundamentals of programming.” (Author's comment: Did this student come to class?)

“Understanding the world of computing beyond programming, and why and how programs and computers actually work.”

“This class is based on computation principles and their limits.”

“It is about the evolution of computing.”

“It's a way to stay away from programming while actually learning something.”

It is too early for us to determine if this course has influenced these students to take additional CS courses since it has only been running for a little more than a year. We will continue to survey students and consult with the other academic schools as we expand the offerings of this course in future semesters.

5. SUMMARY & FUTURE DIRECTIONS

With the need to broaden participation in computing at the high school level, plans are being made to offer this course as an accelerated study summer course for high school students in the local region, with college credit being given for successful completion of the course at this university. We hope to use this course as one way to combat the dramatic decrease in interest in computer science as a major for high school students. [6]

Additionally, future semesters of this course may experiment with other non-traditional programming environments that hide or significantly reduce the need for syntax discussions. Such environments include Alice [3] and Subtext [5].

Readers who are interested in developing and running a course such as the one described here can contact the author for more information and sample course material.

6. REFERENCES

[1] Brin, S. and Page, L. The anatomy of a large-scale hypertextual web search engine. WWW7 / Computer Networks 30(1-7): pages 107-117, 1998. Available at: http://dbpubs.stanford.edu/pub/1998-8

[2] Carlisle, M., Wilson, T., Humphries, J., and Hadfield, S. RAPTOR: a visual programming environment for teaching algorithmic problem solving. In *Proceedings of the 36th SIGCSE technical symposium on computer science education*, pages 176-180, 2005.

[3] Cooper, S., Dann, W., and Pausch, R. Alice: a 3-D tool for introductory programming concepts. In *Journal of Computing in Small Colleges: Proceedings of the 5th CCSC Northeastern Conference*, pages 107-116, 2000.

[4] Denning, P., Horning, J., Parnas, P., and Weinstein, L. Wikipedia risks. *Communications of the ACM* Volume 48 Number 12 (December 2005), page 152.

[5] Edwards, J. Subtext: uncovering the simplicity of programming. *Proceedings of the 20th annual ACM SIGPLAN conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 505-518, 2005.

[6] Foster, Andrea L. Student interest in computer science plummets. *The Chronicle of Higher Education*, Vol. 51, Issue 38 (May 27,2005), page A31.

[7] Gershenfeld, N. and Chuang, I.L. Quantum computing with molecules. *Scientific American*, pages 66-71, June 1998. Available at http://www.media.mit.edu/physics/publications/papers/98.06.sciam/0698gershenfeld.html

[8] Goldman, K. A concepts-first introduction to computer science. In *Proceedings of the 35th SIGCSE technical symposium on computer science education*, pages 432-436, 2004.

[9] Goldstein, S.C., Campbell, J.D., and Mowry, T.C. Programmable matter. *Computer*, Vol. 38, No. 6, pages 99-101, June 2005.

[10] Gousie, M. A robust web programming and graphics course for non-majors. In *Proceedings of the 37th SIGCSE technical symposium on computer science education*, pages 72-76, 2006.

[11] Harel, D. with Feldman, Y. *Algorithmics: The Spirit of Computing, 3rd Edition*, Harlow, England: Pearson Education, 2004.

[12] Hickey, T. Scheme-based web programming as a basis for a CS0 curriculum. In In *Proceedings of the 35th SIGCSE technical symposium on computer science education*, pages 353-357, 2004.

[13] Suarez, J., & Martin, A. (2001). Internet plagiarism: A teacher's combat guide. *Contemporary Issues in Technology and Teacher Education* [Online serial], Vol. 1, Issue 4. Available at http://www.citejournal.org/vol1/iss4/currentpractice/article2.htm

[14] Vail, D. and Veloso, M. Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, A. Schultz, L. Parker, and F. Schneider (eds.), Volume II, pages 87-100. Kluwer Academic Publishers, 2003.

[15] Von Ahn, L. Games with a purpose. *Computer*, Vol. 39, No. 6, pages 92-94, June 2006.

[16] Wing, J. Viewpoint: computational thinking. *Communications of the ACM*. Vol. 49, Issue 3 (March 2006), pages 33-35.