Adding Computer Science to an Introductory Computing Class for Non-Majors

Gail Carmichael
Carleton University
1125 Colonel By Drive
Ottawa, Canada
gbanaszk@connect.carleton.ca

ABSTRACT

This classroom experience report outlines changes made to the curriculum of a computing class for arts and social science students that has only taught common software usage in the past. Survey results show that students liked the addition of computer science topics including algorithms, human-computer interaction, and visual programming with the Scratch environment.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Curriculum

General Terms

Algorithms, Human Factors

Keywords

non-majors, arts and social sciences, curriculum, Scratch, CS Unplugged

1. INTRODUCTION

The School of Computer Science at Carleton University offers an introductory course for non-majors called COMP 1001: Introduction to Computers for Arts and Social Sciences. The course is taken as a science elective. The undergraduate calendar [1] provides the following description:

> This course is intended to give students in the arts and social sciences a working knowledge of computers and their applications; computer fundamentals; use of computing facilities; introduction to graphical user interfaces; a sampling of software packages applied to problems in the arts and social sciences.

Based on this, the course content has typically covered advanced use of software packages that students studying arts and social sciences would find useful. For example, the Microsoft Office suite is usually the main focus. Some basics of data representation, including binary numbers, are also touched upon.

In preparing to teach this course for the summer 2010 term, I considered the opportunity to introduce the students to computer science in a way that would be relevant to their own studies. By doing so, they could gain an appreciation for the field, exercise their mind by practicing computational thinking, and find out how computer science can help them accomplish their goals. There was also the opportunity to bring the course up to date, since most students seem to have basic computer knowledge and abilities today.

Others have tackled the issue of including computation in courses for non-majors. For instance, Joyce [12] reported on a popular course that focused on how people solve problems with computers, where the first assignment involved writing an essay on the topic. Rodger [18] described a course for non-majors that introduced computer science topics by having students make scripts and program for animations and virtual worlds. Guzdial [10] used computation for communication as a guiding principle in designing a programming course for non-majors. He and Forte later reported on the design process and measured success for that course [11].

Unlike these examples of courses for non-majors, my aim was not to concentrate solely on programming, or computers as a problem solving tool. Instead, I combined these topics with other areas of computer science so that every student would have the chance to connect with some aspect of the course. This paper summarizes my experience teaching the new curriculum. A review of the course content as it was previously taught begins Section 2, followed by details of the reinvented curriculum. Results of a survey completed by students in the course can be found in Section 3 along with suggestions for improving the new curriculum before it is taught again.

2. COURSE CONTENT

In this section, I will outline what the course looked like the semester before I taught it, based on the notes and assignments kindly provided to me by the instructor. I will then discuss my reinvented curriculum, including my objectives, the topics I covered, and how I assessed the students.

It is important to note that because I taught this course in the summer, the term was condensed into 6 weeks, where each week had two three-hour classes instead of one. I also had a much smaller enrollment of sixty students instead of the several hundred that register during fall and winter

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.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

terms. Summer courses also do not have formally scheduled tutorials, while the courses during the fall and winter terms often include one hour of tutorial per week.

2.1 Previous Curriculum

The topics covered in the past were:

1. **Introduction to Computers**. An overview of computer system basics, from hardware components to commonly used software.
2. **Data Representation**. How to count in binary and represent text and images with binary digits. How files and folders work.
3. **Introduction to Computer Software**. General introduction to operating systems and software.
4. **Introduction to Windows**. History of Windows and basic usage of Windows Vista.
5. **Introduction to the Internet**. What the Internet is including history and useful terminology. How to write basic HTML.
6. **Introduction to Microsoft Word**. Basic and advanced use of Word functionality.
7. **Introduction to Microsoft PowerPoint**. Basic and advanced use of PowerPoint functionality.
8. **Introduction to Microsoft Excel**. Basic and advanced use of Excel functionality.
9. **Introduction to Databases**. Overview of how databases work, and how to create them in Microsoft Access.
10. **Introduction to Open Source Software**. Overview of licensing and philosophy, and a selection of OSS applications.

There were four assignments for the course, ten tutorials to be done during scheduled lab time, a midterm, and a final exam. The first assignment contained short answer questions on basic computer usage and data representation. The second assignment required students to write a web page in HTML. The third assignment asked students to format existing text in Microsoft Word. The last assignment was to create a budget in Excel.

2.2 Reinvented Curriculum

2.2.1 Objectives

My goal with the revised curriculum was to continue exposing students to a variety of software they would find useful in the future while introducing them to a selection of computer science topics. Based on this, the three main course objectives were to:

1. Gain an appreciation of what computer science is and how it relates to the arts and social sciences.
2. Practice computational thinking.
3. Learn about the software and tools that will help you succeed in an undergraduate program.

2.2.2 Course Topics

I chose topics that linked back to the course objectives, and attempted to include the most useful material from the existing curriculum. I also drew inspiration from my previous experience creating and teaching a week long mini-course on games and computer science for grade eight girls [7]. I was able to use some of the same material adapted to the undergraduate level. I included many interactive classroom activities in my lectures to keep the students more engaged and encourage a deeper understanding of the subject.

The topics in the order I taught them were:

1. **Introduction to Computer Science and Computational Thinking**. A broad sense of what computer science is and how it connects to various other fields.
2. **Binary Numbers and Data Representation**. Binary counting, text representation and compression, and vector and bitmap image representation.
3. **Basic Programming with Scratch**. Key programming concepts including loops, conditionals, and variables.
4. **Introduction to Microsoft Word and PowerPoint**. Basic and advanced use of Word and PowerPoint.
5. **Algorithms**. Linear, binary, and hash table search, selection sort, and quick-sort.
6. **Human Computer Interaction**. Application of select principles and theories to existing projects.
7. **Using the Internet Effectively: Security, Software and Tools**. Searching, website creation, wikis, etc.
8. **Open Source Software**. Licenses, philosophy, common OSS applications.

When introducing computer science, I played a video produced by the University of Washington that shows five computer scientists working in various types of jobs and research [20]. The video gives a good sense of what computer science is, and most who watch find they can connect with at least one of the people featured. This is particularly beneficial to women, who tend to attach their interest in computer science to other areas of life [15]. We discussed in more depth how computer science related to many of the fields the students were studying, such as psychology and law.

For the data representation topic, I created poster cards representing binary digits from the CS Unplugged [6] binary numbers activity, and asked volunteers to use them to count at the front of the classroom. We looked at how ASCII codes worked, and used the CS Unplugged activity on text compression to see how data might be represented in a more compact way. We also saw the difference between bitmap and vector graphics and discussed their use in 3D graphics.

I spent one three-hour class giving a tutorial on Scratch [9], encouraging the students to try a few tutorials on their own. We looked at some of the content in lesson plans from The Irish Software Engineering Research Centre [19] and videos from Learn Scratch [3]. I walked through how to create a simple game as an example of what they might want to make on their own.

I limited my instruction on Word and PowerPoint to two three-hour classes, and based the content almost completely on the previous year's lecture notes. Instead of lecturing, however, I created tutorial sheets that I handed out to students. Each volunteer used their tutorial to teach a small number of techniques to the rest of the class.

My lesson on algorithms was largely intended to provide a concrete opportunity to practice computational thinking. I once again adapted CS Unplugged activities on sorting and searching to allow the students to discover how these algorithms worked experientially. For sorting, I gave eight volunteers a random piece of paper with a number marked on it. A ninth volunteer had to sort the others based on the rule that she could only ask two volunteers who had the bigger number. The students worked in pairs to try linear, binary, and hash table searching using the same set of numbers found in the CS Unplugged searching activity.

In the human-computer interaction class, I introduced the general idea, then described several theories and principles used by designers. In particular, we looked at Donald Norman's principles of design and seven stages of action from his book Design of Everyday Things [17]. Then I played Norman's TED talk on emotional design, based on his book on the same topic [16]. I also briefly described the concepts of embodied and situated cognition and how they apply to learning, using two augmented reality projects as examples (Environmental Detectives [14] and Construct3D [13]). The students then watched several videos of up and coming interfaces and discussed how these theories and principles could apply.

For the lecture on the Internet, I asked a PhD student in security to give a guest talk on her research. I showed some tips for searching effectively online, contributing to wikis like Wikipedia, and other related topics. HTML was not covered in depth; instead, I gave a brief overview of the syntax and available resources. I also demonstrated how to make a web site with tools like Google Sites [2] and suggested a few online tools the students would find useful for collaborating on school projects.

Finally, I introduced open source software (OSS). To understand the idea behind OSS licensing, we looked at Creative Commons licenses [8]. We discussed the pros and cons of making your work available with such licenses, and then I showed some freely available open source software the students might want to use in the future.

2.2.3 Assessment

Due to the compressed nature of the term, I opted to have two assignments and a four-part project, but no midterm. The final exam tested students' understanding of most of the material discussed in class, though details about how to use software were not covered.

The first assignment was given in the first week of class. It was called "How Can Computing Help?" and was intended to get students thinking about how computer science is related to something they care about. Students were asked to think of a hobby or their own field of study, and to research how computing can make tasks in that hobby or field easier. For example, a photographer might discuss how searching algorithms make it easier to find tagged photos in a large collection, and they might mention some image processing techniques found in image editing software. In the process, they learned how to search for sources related to computer science, and saw the kind of issues they could approach a computer scientist with if they ever needed to collaborate with one. The result was a short paper of at least 1000 words.

The second assignment was to create a project (or several smaller projects) in Scratch. Students were free to create anything they wanted to, from an interactive animation to a simple game. Their grade was based on a set of technical requirements, such as using at least one loop, if statement, and variable. They also had to incorporate some kind of user interaction. The students were invited to show their projects in class so they could feel pride in their creations.

The project was a set of four assignments related to a particular piece of software each student chose to learn about. There was a limit on how many people could work on a single piece of software, and software shown during lectures was off-limits, so variety was guaranteed. Each part of the project was designed to provide an opportunity to learn advanced functionality of Word, PowerPoint, and a website building tool for a real and practical purpose.

In the first project assignment, students were asked to write a tutorial about their software. Again they had considerable freedom to choose what and how much to write, and were given a set of technical requirements related to advanced formatting in Microsoft Word. The idea was that by giving students had a real purpose for using the formatting instead of simply applying it to previously written text, they would feel more motivated.

In the second project assignment, each student exchanged their written tutorial with another student, and used Word's change tracking feature to make comments, corrections, and suggestions. The original tutorials were not graded until students received the reviewed version and had a chance to improve it. The idea was to provide feedback before assessment to help students produce the best work possible (something emphasized in Bain's research of the best college teachers [5]), and to expose the reviewer to a new piece of software in the tutorial they were reviewing.

The next part of the project involved forming small groups and presenting software that someone in the group had written about. This gave students the opportunity to learn about even more software other students had studied, create a presentation with PowerPoint, and practice effective presentation skills.

The last project task was to transform the written tutorial into a web page. Students were encouraged to use either Google Sites [2] or Weebly [4]. An interactive component was required, such as a click-through tutorial built with screenshots in Scratch.

3. EVALUATION AND OUTCOME

At the end of the term, I asked students to fill in an anonymous, informal survey online to assess the success of the course. Out of the 61 registered students, 26 completed the survey. Most completed it before the final exam. This section summarizes some of the results, followed by suggestions for improvement.

3.1 Survey Results

The questions in the survey were designed to capture general attitudes of the students taking this course as well as determine how successful each component of the course was. The first set of questions provided a statement and a scale

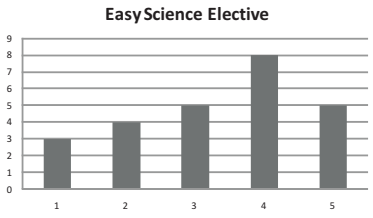

Figure 1: Survey results for "I took this course because it was an easy science elective."

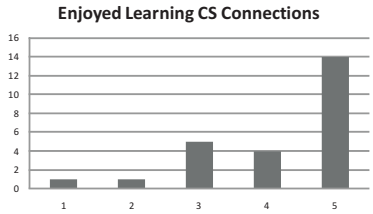

Figure 2: Survey results for "I enjoyed learning what computer science is and how it connects to other fields."

from 1 (strongly disagree) to 5 (strongly agree). There were several short answer questions at the end.

As this course had a reputation of being an easy elective, the first question was used to determine whether that was the motive of the student for taking it. The results are shown in Figure 1. Just over half the respondents said they agreed or strongly agreed with this statement and 5 were neutral. Because so many were willing to admit that this was their reason for signing up, it would be easy to suggest that they were not interested in learning more about computer science. However, based on the remaining survey results, this was not the case.

For instance, despite the fact that 15 students agreed or strongly agreed that they were worried about the new course content, 21 agreed or strongly agreed that they were looking forward to learning something new. Perhaps a more important indicator is the response to the statement on enjoying learning about computer science and its connections to other fields, summarized in Figure 2: 19 students agreed or strongly agreed that they enjoyed this topic. In fact, more than half the respondents agreed or strongly agreed that they enjoyed learning about each topic in the course, except for searching and sorting algorithms. Only 8 agreed or strongly agreed they enjoyed learning about algorithms, and 12 were neutral.

For each topic in the course, students were given a statement that said they felt they understood it. The results for binary numbers and data representation are in Figure 3, for Scratch in Figure 4, for algorithms in Figure 5, and for human-computer interaction in Figure 6. These are the main topics that covered actual computer science concepts and thus give the most insight into the success of adding computer science concepts into the curriculum. More than half the respondents said they understood each topic except for searching and sorting, where 12 agreed or strongly agreed they understood it. Based on this, it is fair to conclude that non-major students without a previously avowed interest in computer science are capable of understanding computer science topics if they are presented in an effective way.

The portion of the survey that allowed students to write short answer questions gives a good idea of what specifically worked well in terms of teaching methods and content, and what needs to be improved. One of these questions asked what piece of software or topic the respondent found the most interesting, without giving any options to choose from. Some students listed more than one topic. The topics noted were: Scratch (9), software from student projects (5), human-computer interaction (3), Internet security (2), none (2), all topics (2), binary numbers (2), and open source

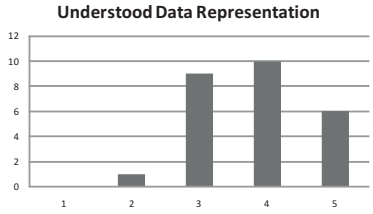

Figure 3: Survey results for "I feel I understand data representation (binary numbers, images, etc)."

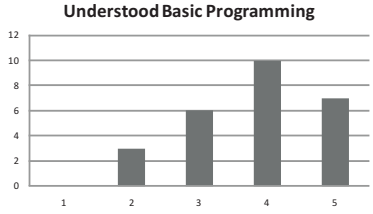

Figure 4: Survey results for "I feel I understand basic programming concepts after using Scratch."

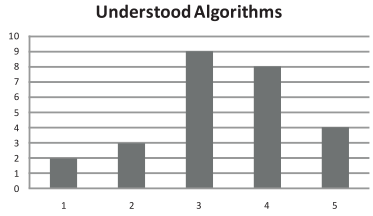

Figure 5: Survey results for "I feel I understand searching and sorting algorithms."

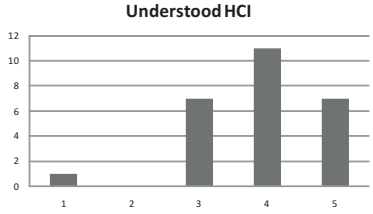

Figure 6: Survey results for "I feel I understand the ideas behind human computer interaction."

software (1).

When asked about the assignments specifically, some students felt that the Scratch project was interesting but not relevant to their own studies. Some said there was too much work for a summer course while others suggested the assignments were easy. Several students said that learning the more advanced functionality of Word for the assignments was one of the most useful things they will take away from the course. A selection of both positive and negative comments written by students about the course topic and assignments follows:

- Using the review, formatting and reference tools in Word is something that I could have used a lot in the last three years to make my essay writing more efficient.
- I never dreamed I could program my own computer game and want to put it on my website for my field of study!
- I thought scratch was a really nice and easy program to use to let your imagination run wild :)
- Perhaps a little more focus on HCI I think a lot of people will find this relates more to their arts/social science degree, and got a little lost when we were covering this material.
- Making the website was a great idea. Very useful. The first assignment was useful. It helped look at academic sources (articles) and gave me an idea of what research is out there in the compsci field.
- Scratch was an annoying, difficult program and will bear no usefullness to me in the future.
- I would discard the Scratch assignment and focus on more academic type programs.

Some students found the presentations useful, especially because I gave them specific tips on how to give an effective talk. Unfortunately, there was not really enough time to have presentations that covered software in any detail. I asked students to give an overview instead so the rest of the class would know whether the software would be useful to them. Some specific comments about the presentations follow:

- Very helpful. Also, I found some of the other presentations to be very helpful as well.
- I learned about several software programs that I have never heard about before.
- It was hard to sit through things that I already knew about.
- While useful to an extent, not having the presentations for future use limits their usefulness
- The presentations were very basic, in fact not even tutorials (more selling the software) so we didn't get a chance to learn a lot about the actual software programs. It was interesting though to hear about different programs.
- I did find it useful to learn a new piece of software but found the presentations were not as informative.

Much of the content was delivered via activities and discussions. This was appreciated by many students who found it was easier to pay attention and learn the content in a deeper way. Many specifically said they appreciated the binary numbers demo. A small number of respondents very much disliked this style of teaching, however, saying that the activities were juvenile. Some of the comments about the activities follow:

- Overall I feel that all activities helped us get a more hands on learning experience, and helped the 3 hour lecture go by a little quicker.
- Binary numbers making the 8 charts and have students come up and sort through the numbers was useful.
- The exercise understanding binary was something I will never forget for the rest of my life. Because it was a physical/visual exercise i will always remember how to convert binary to decimal.
- i hated all the class participation. it made me feel like i was 5
- The classroom exercises seemed a little childish.

Finally, a selection of comments about the course in general follows:

- Next to the overall usefulness/value of my psychology program, this is by far the most influential course of my undergraduate career.
- I liked learning more about how computers work and function. As an arts major it is difficult to get into most computer science courses.
- Overall the course and materials were very well presented and will no doubt be more helpful than continuing with the previous course content.
- Going in, at this day and age you tend to think you already know about computers and how to use some of the softwares that are listed in the course but you soon realize that there is a lot you don't know.

In summary, while there are some aspects to improve, the course was well received overall. The students appreciated being challenged and learning about real computer science, and they felt confident about their knowledge and understanding of the subject.

3.2 Future Improvement

While the new course content has been a success overall, there are several areas for potential improvement. The following suggestions are based on my own reflection and observation as well as the survey results summarized above.

The order of the topics covered may not have been ideal. For instance, an introduction to Word and PowerPoint should probably be the first topic after the introduction so that students can use the built-in reference system on the "How Can Computing Help" assignment. This assignment might also be better situated after the lecture on human-computer interaction to give more ideas for connecting the topic of choice to computer science. Algorithms could be taught before Scratch to put students in the mindset of computational thinking before attempting to create their programs, but

learning Scratch first might help make the case for sitting through a class on algorithms, since it may be seen as more fun.

I would not change any of the main topics for the course. As expected, some students did not like learning how to program, but a surprising number of them were very positive about it. It is not necessary to use Scratch, but some kind of visual programming language is probably a good idea since it takes away the overhead of learning how to compile and allows for quick and easy visual output.

The utility of the presentations was poor compared to the time spent on them, and they would be impractical for a large class. Instead, the written tutorials could be shared with all students. Advanced use of Excel, as taught in the previous curriculum, would be a more relevant skill for this course.

I would suggest continuing to use interactive activities as much as possible in the future. Some appear to be more difficult to use in a large classroom, but can either be done as demonstrations by volunteers at the front of the room (as I did with the sorting activity), or adapted to be completed on laptops brought to class by students.

4. CONCLUSION

This paper reported on a new curriculum for a computing class taken as a science elective by arts and social science students. A survey completed by the first students the curriculum was taught to showed a positive attitude toward the addition of computer science topics to a course that previously focused on software usage. The use of interactive classroom activities was also mostly well received.

These results show that computer science concepts can be made accessible for students not majoring in the subject. Despite the fact that many may be enrolling simply to obtain their science elective credit, and thus do not have a deep interest in learning computer science, they are able to learn and enjoy the topic when it is presented to them effectively. The key factor appears to be connecting computer science to their own fields of study and interest while providing them with useful skills they currently lack (such as advanced word processing and web site creation).

Computer science departments that have similar courses can feel confident in taking a chance and updating the curriculum to include more technical topics. As an added bonus, perhaps some non-majors will even discover a love of computer science and find themselves looking for a new minor.

5. REFERENCES

[1] Carleton University Undergraduate Calendar. http://www.carleton.ca/calendars/ugrad/current/ [accessed September 2010].

[2] Google sites. https://sites.google.com [accessed September 2010].

[3] Learn scratch. http://learnscratch.org [accessed September 2010].

[4] Weebly. http://www.weebly.com/ [accessed September 2010].

[5] K. Bain. *What the Best College Teachers Do*. Harvard University Press, 2004.

[6] T. Bell, I. Whitten, and M. Powell. Computer science unplugged. http://csunplugged.org/ [accessed September 2010].

[7] G. Carmichael. Girls, computer science, and games. *SIGCSE Bull.*, 40(4):107–110, 2008.

[8] C. Commons. About licenses. http://creativecommons.org/about/licenses/ [accessed September 2010].

[9] L. K. Group. Scratch. http://scratch.mit.edu/ [accessed September 2010].

[10] M. Guzdial. A media computation course for non-majors. *SIGCSE Bull.*, 35(3):104–108, 2003.

[11] M. Guzdial and A. Forte. Design process for a non-majors computing course. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 361–365, New York, NY, USA, 2005. ACM.

[12] D. Joyce. The computer as a problem solving tool: a unifying view for a non-majors course. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 63–67, New York, NY, USA, 1998. ACM.

[13] H. Kaufmann, D. Schmalstieg, and M. Wagner. Construct3d: A virtual reality application for mathematics and geometry education. *Education and Information Technologies*, 5:263–276, 2000.

[14] E. Klopfer and K. Squire. Environmental detectives - the development of an augmented reality platform for environmental simulations. *Educational Technology Research and Development*, 56:203–228, 2008.

[15] J. Margolis, A. Fisher, and F. Miller. Caring about connections: gender and computing. *Technology and Society Magazine, IEEE*, 18:13–20, 1999.

[16] D. Norman. *Emotional Design: Why We Love (or Hate) Everyday Things*. Basic Books, 2003.

[17] D. A. Norman. *The Design of Everyday Things*. Basic Books, September 2002.

[18] S. H. Rodger. Introducing computer science through animation and virtual worlds. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 186–190, New York, NY, USA, 2002. ACM.

[19] The Irish Software Engineering Research Centre. Scratch lesson plans. http://www.lero.ie/educationoutreach/secondlevel/ [accessed September 2010].

[20] University of Washington. Why Choose CSE? http://www.cs.washington.edu/WhyCSE [accessed September 2010].