Why Computer Science Matters: Introducing Computer Science using Scratch and TouchDevelop

Suresh K. Lodha
University of California
Santa Cruz
lodha@soe.ucsc.edu

Sonali Somyalipi
University of California
Santa Cruz
sonali@soe.ucsc.edu

ABSTRACT

Motivated to emphasize the importance of computer science to society in an introductory computer science class and to bring out the talents of a diverse group of learners, we utilized TouchDevelop to allow experimentation with various sensors on mobile platforms and Scratch for creative interactive story telling. In addition to the traditional emphasis on concepts, algorithms, hardware, and programming language constructs, emphasis was placed on exposing and inviting students to participate in answering the following question: Why Computer Science Matters? This paper presents some sample final projects using Scratch and Touchdevelop as well as a brief summary of students' efforts to expound on the importance of computer science to society. Additional challenges of devising a flexible grading rubric to meet the needs of the diverse student population are also discussed. Pre and post course surveys were administered to measure the impact. Although the overall rating of the course as a learning experience decreased slightly from a prior rendering of the course by the same instructor using C++, 33% of students indicated that they are more likely to take additional computer science courses as compared to 21% who are less likely. Further experimentation and assessment is needed to evaluate a programming vs. non-programing introduction to computer science. For a programming-based introduction, streamlining of websites and grading rubric coupled with a flexible learning experience to meet the diverse needs and expectations utilizing some combination of Scratch, TouchDevelop, C++, Processing or Python is likely to yield better results although at the cost of requiring additional teaching resources.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: [Curriculum, Literacy, Self-assessment]

General Terms

Human Factors, Languages, Experimentation

Keywords

Computer science education, computational thinking, mobile devices, TouchDevelop, Scratch, non-majors, creativity support tools, Twine

1. INTRODUCTION

The Department of Computer Science at the University of California Santa Cruz has traditionally offered CMPS 10, *An Introduction to Computer Science* class intended for non-majors. The class has primarily been taught using a combination of topics found in a traditional introductory book such as *An Invitation to Computer Science* or *Computer Science Illuminated* supplemented with programming exercises using C++. The typical algorithms covered in the class include searching (sequential and binary search) and sorting (selection, insertion, and quicksort) algorithms. Typical programming projects revolve around implementing some aspects of these algorithms. The class has been taught using this methodology for the last ten years more or less with little change. Roughly two-thirds of the students have consistently rated the course in the category of very good or excellent combined as an overall learning experience. Given the ratings in lower division undergraduate course at our institution, these ratings are acceptable not requiring any urgent need to change or alter the course.

Yet the movements inspired by computational thinking and AP CS principles course point the direction to deliver a more compelling rendering of the course that motivates students to appreciate the evolving role of computer science and its importance in solving problems of interest to society. This theme also emerges in conversations with the undergraduate students who are much more interested in media and mobile platforms. It appears that many students merely tolerate this course but would be delighted to have a more modern version that makes the subject interesting and relevant.

Guided by this thinking, in recent years, some of our colleagues in the department has been championing different renderings of the same course with perhaps emphases on different topics or different programming languages. These efforts have resulted in at least five different renderings of the course in last one and a half years by four different instructors. In addition to the traditional approach described before, one instructor chose to present a wider variety of computer science topics at a higher level often using historical contexts and by creating compelling powerpoint presentations. This was essentially a non-programming introduction to computer science with only two programming tasks assigned – a drawing program using high level context-free drawing software and an introductory programming assignment using Scratch. Students reported a very high level of satisfaction with this rendering of the course with 82 percent evaluating the course in the category of very good or excellent combined for its overall learning experience. This finding is consistent with the findings by Cortina [6] who reported increased enrollment and interest towards a non-programming introductory computer science course at CMU and those by Lee [13] who also reported success in offering an introductory CS course without programming component at Cornell University.

In spite of the great success of this model, subsequent renderings of the course introduced different or additional programming languages as essential vehicles to introduce computer science. One of the these renderings utilized *processing* as the language, the second utilized a combination of C++ and Scratch, and the third one offered a *choice* between Scratch and TouchDevelop. This paper focuses on describing the approach and outcomes of this final approach employed recently.

2. BACKGROUND

Most disciplines require an introductory computer science course as a part of general education. Many students take this course because they are required to. Many others want to understand the utility and relevance of computer science to disciplines that they are pursuing. But almost everyone wants to engage with emerging gadgets and media to have fun and for social networking. They also want to use it on the run, that is, when they are mobile, and in between tasks. Major challenge in a traditional rendering of a computer science course is that the preciseness and detail needed to write computer programs correctly can be overwhelming to the students and often distracting. Students are also mostly unaware of the vast array of emerging applications to society (such as Kiva, KickStarter, theCausemopolitan) that computers have enabled through crowd-sourcing and mobile applications. We have only one quarter to engage students and endear them to computer science or empower them with technology. How can we do this?

First, there are opportunities to introduce computer science without the use of programming languages. As far back as in 2002, Lee [13] reported positive experience in introducing computer science using natural language processing, information retrieval, and artificial intelligence at Cornell University. More recently, Cortina [6] introduced computer science to undergraduate students utilizing a flowchart simulator Raptor rather than any programming language and reported increased enrollment (although still with small class sizes of around 40 as compared to our class size of 250) and interest both from students and departments.

Then, there are several quite successful efforts using *tailored* courses with focus on one specific sub-discipline. Forte and Guzdial [9] reported encouraging outcomes with tailored course on media computation at Georgia Institute of Technology. Biology-themed courses have been popular at Harvey Mudd College and Williams College [14]. An introductory computer science course focusing on artificial intelligence is also offered at Williams College. Heavily tailored courses such as introductory courses on computer graphics, animation and virtual worlds, (or computer gaming) are also being offered at UCSC and other institutions [16]. These courses do not belong to similar category of courses such as introductory computer science courses, and are typically not meant to replace them.

Also, introductory computer science courses for engineers are offered at several institutions [9, 10] and are not meant as general education courses for students majoring in arts, humanities, and social sciences. Although there were some early efforts to offer introductory computer science courses for artists and social scientists at UCSC, those courses are no longer offered.

A strong need for general purpose introductory computer science courses for broader audiences, say for students with non-declared majors, or those likely to major in social sciences, arts, and humanities remain. The question remains: what do students want to learn? How to keep them engaged and motivated?

2.1 Computational Thinking

Since Jeannette Wing [22] coined the term *Computational Thinking* in 2006 arguing that it is comparable to mathematical, linguistic, and logical reasoning that must be taught to all children, the movement to introduce computer science in ways other than a traditional approach has gained momentum. As far back as in 1998, Joyce [11] and Ben-Ari [3] advocated teaching computer science as a problem-solving tool. A recent New York Times article [17] states that many professors at various colleges including CMU, Wheaton College, University of Maryland (Baltimore), Rutgers University, and Grinell College (Iowa) are introducing courses to teach computational thinking to students. Hambrusch et al. [10] use Python to teach computational thinking to non-majors. Innovation by LeBlanc [17] in teaching computer science to poets by analyzing large bodies of text and by breaking large problems into small ones is certainly inspiring.

What is computational thinking? Computational thinking is the mental activity in formulating a problem to admit a computational solution that can be carried out by a human or a machine. The greatest emphasis in computational thinking is on the abstraction process. Two workshops [7, 8] on computational thinking have begun to define the nature and scope of computational thinking and address pedagogical aspects. Researchers are addressing the question as to what is the right age to introduce computational thinking in a curriculum – middle school, high school, or at the college level? The College Board, in collaboration with NSF, is designing a new AP (Advanced Placement) course (in addition to the existing AP computer science course) that covers the fundamental concepts of computing and computational thinking (see www.csprinciples.org). Five universities – University of North Carolina at Charlotte, University of California at Berkeley (UCB), Metropolitan State College of Denver, University of California at San Diego, and University of Washington at Seattle (UW)– have piloted different versions of computational thinking courses. Of these, UW is using *processing* as the programming language and UCB is using a combination of Scratch and BYOB. Google has launched a website (www.google.com/edu/computational-thinking) on Computational Thinking and Computer Science (www.csunplugged.org) unplugged teaches computer science without the use of a computer.

How should this revolution in thinking about basic computer skills ought to impact introductory computer science courses at college level? Is programming language essential for teaching an introductory course to non-majors? How does computational thinking relate to students' need for engagement and enrichment with gadgets, media, and mobility and impact of computers on society [5]?

2.2 Scratch Programming

Several institutions have adopted Scratch as one of the programming languages [4] used early in an introductory computer science course including courses at UCB and Harvard. There is also a robust discussion of pros and cons of using alternatives such as Alice and Greenfoot [20].

We decided to use Scratch in combination with C++ in an experimental version during Summer 2011 with a small class size of 35 students. Students provided highly positive feedback because it allowed them to learn the programming language rather quickly and captured their creativity. Buoyed by this experience, in our current offering, we decided to use Scratch as the only programming language for a significant majority of students in a class of 140 students.

The Scratch programming language developed at the MIT Media Lab offers a visually appealing environment allowing novice programmers to learn programming without initially having to write syntactically correct code. Users can create games and interactive stories that are also animated. Scratch programming depends on the inquisitiveness of the learner and his/her capacity to explore and play with the programming constructs. In the spirit of this philosophy, the Scratch language is based on programming 2-D graphical objects called sprites, set against a background called the stage. Students write scripts with graphical blocks that represent various programming constructs to animate the sprites, make them interact amongst themselves, and change their appearances. The Scratch application allows users to import images and sounds, apart from creating their own media, to make media-rich projects.

Scratch is clearly a programming language for beginners as it has a minimal set of language blocks and command set with an easy to use application interface. The color-coded commands are categorized by their functionality and are presented in panes on the same window. This way the users are always looking at the commands and hence it is highly probable that they can enhance their projects better by using additional commands and constructs.

By not having distinct compile and edit phases, Scratch programming is easier for users as they can change their program while it is running and can test program fragments to see what they do.

Scratch uses blocks which fit into each other like toy building-bricks, only when their combination is meaningful and right. This way Scratch does not have to post error messages. There are several other mechanisms which make understanding of constructs better and makes errors much less probable. For instance, variables in Scratch are concrete and visible to the users. The users can see the effects of any commands they execute on variables. Also, the language has just three data types which can be easily identified by the shapes of parameter slots and function block shapes. Thus, incorrect programs do not throw errors, but they do run, behaving in unexpected ways though. This encourages the user to think carefully and write programs that behave as he expects them to.

Scratch programming also encourages sharing and learning across peers. The Scratch Website allows easy sharing of projects, that makes collaboration and receiving feedback simple. Scratch also supports the 21st century learning skills: information and communication skills, thinking and problem-solving skills, and interpersonal and self-directional skills [2].

2.3 TouchDevelop: Programming on a Phone

With the widespread use of smartphones, and their augmented processing power, battery life and screen resolutions, mobile devices like smartphones and tablets are increasingly becoming the devices of choice for personal computing as opposed to ordinary laptops and PCs. It is interesting to see how mobile devices themselves can be used as platforms to develop applications targeted to run on them. This will encourage users to write scripts for themselves to automate their tasks, or write small applications of their choice for fun and productivity. Interestingly, this interface being more exciting can also be used to invite beginners to learn programming, as the results are more immediate,exciting and practical.

The TouchDevelop language developed by Microsoft for its Windows Phone[19] is such an environment, useful in developing applications and scripts, to be run on the Microsoft Windows smartphones themselves. The only way to author code in this language is via the touchscreen based code editor in the Windows phone and the resulting programs run on the Windows phone with full access to the phone's media, sensors and cloud connectivity to storage and social networking. The phone's code editor provides touchscreen buttons for the language's syntax elements and built-in primitives that in turn allow access to the phone's media and sensors.

The TouchDevelop language which has properties drawn from imperative, object-oriented and functional languages, is a statically typed programming language. Scripts written in the TouchDevelop language are composed of procedures which are termed as "actions" and have global variables. There are also event-handler like elements which bind events like button clicks, screen swipes, and screen taps to actions. Actions are made up of statements, and expressions, where expressions in turn are composed of literals and various operators. Actions can also have local variables, invocations to other actions, and multiple return values.

There are several data types in TouchDevelop and each type exposes a set of properties, that can be accessed from within actions. However, in order to keep programming simple for beginners, there are some language limitations. Some examples of such limitations are the absence of error-handling and user-defined types.

3. COURSE OVERVIEW

The course began on a inquisitorial note seeking answers from students as to "Why" certain fields of study like Astronomy needed to be pursued. The answers unequivocal and quite uniform. Astronomy helps us understand the mysteries of the universe. Whereas answers to a parallel question - "Why the study of computer science needed to be pursued?" proliferated into a wide array of correct but incomplete responses. Most responses focused on stating the usefulness of computers in hosting websites or social networking or telecommunications. Before proceeding into the aspects of what the tenets of computer science are or how these are realized as artifacts of say hardware or software, the students were drawn to answer the question as to "Why computer science mattered?". This methodology is also supported by the idea of teaching computer science in context as discussed in detail by Cooper et.al [5]

The definition of computing as put forward by the AP College Board, was used as the starting point for a discussion about why pursuing the study of computer science mattered. The definition read as follows: "Computing is the development and use of computers and computer programs to solve problems of importance to humans". Not a single response in a pre-course survey came anywhere close to this definition of computer science. Students were then asked to identify problems of importance to humans that could or already were being solved by the application of computer science. This question elicited responses surrounding the medical and robotic applications mostly. Several examples of cutting-edge entrepreneurial efforts including kiva, kickstarter, and theCausemopolitan, were presented. Compelling applications of computers to society in various aspects encompassing health, education, environment, entrepreneurship, social causes, global poverty were presented and discussed. Students were assigned the task of creating a one page powerpoint presentation listing 3-5 applications of computer science providing concrete examples, citing sources, and describing briefly how computers were useful in the specific applications cited. This effort was continually kept up via the story building assignments delegated to the students throughout the course. In story telling effort, these applications were to be contextualized in the form of a real or imagined individual who benefited from this application and how it altered or impacted her or his life.

Gradually, students were introduced to a more granular definition of computing which said "Computing skills include the ability to write programs, represent and manipulate data/data sets, and to give directions to make a computer do what you want". This definition lays down the foundations for teaching a novice what programming is and why programs are necessary artifacts in computer science.

The next discussion was about how solving problems using computers required communicating the known solution to the computer as one would to a child in middle-school. This analogy was used primarily to introduce the notion of pseudocode.

Abstraction was realized as an important pedagogical tool in the lectures as good pedagogy requires the instructor to keep initial facts, models, and rules simple and only expand and refine them as the student gains experience[23]. So, easy to understand analogies were often used to convey essential programming constructs during the lectures. For eg. Variables were as traditionally done, compared to boxes, and using this model students were taught variable naming conventions, the notion of variables storing values, and semantics of an assignment operation. In parallel, the idea of variables being available in more than one type, namely -as numbers, logical variables and character strings was also made known, with the information of what basic operations each of these variable types supported.

For purposes of simplicity and in the interest of abstraction, pseudocode written was always meant to be read in sequence. In later lectures, students were introduced to conditional statements like IF-THEN-ELSE. Looping constructs like FOR and DO-WHILE were also taught as means of performing repeated tasks without having to write redundant code. With these students were also instructed to use techniques like tracing pseudocode in order to understand and comprehend given snippets of pseudocode. Good programming practices like commenting code were also taught. Cryptic smart programs, for example repeated subtraction to compute GCD, without appropriate comments, could be a burden rather than an asset. Additional components of the class are described next.

3.1 Story Telling Using Twine

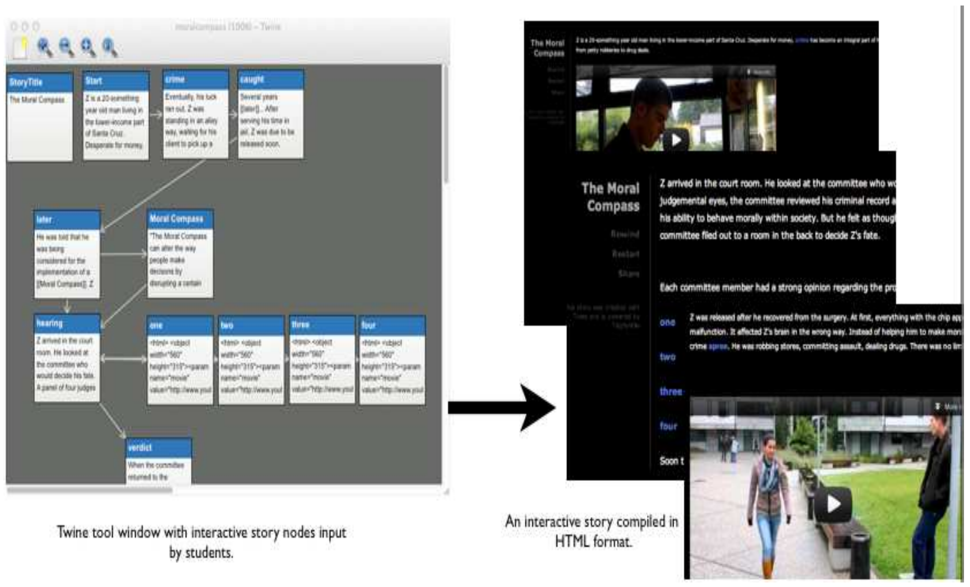
Interactive-text based stories are not only a good medium to bring out students' creativity and logical thinking, but also are a good way to stretch their imagination and knowledge. Since, the students were required to build stories whose central theme was - "Computer Science", they were encouraged to look up and learn the history and advances of computer science in order to build a strong plot. Twine[12] is an easy to use desktop utility to create interactive texts, that students used during the course to create and compile their stories. The software enabled them to organize their story graphically in a node based flowchart. Here, each piece of text is represented within a node, and lines link these nodes, showing the routes the reader might take through the story. This flowchart representation could be exported and compiled into a a single web page. Twine also allowed inclusion of conditional text. Figure 1 shows a snapshot of a sample Twine window with a student's story in the flowchart format on the left and in the web-page format on the right.

3.2 Programming Assignments in Scratch

The Scratch programming assignments were modeled as interactive story scenes. The students were required to create an animation sequence with appropriate objects(sprites), backgrounds and sound clips. The assignments were modeled in such a way that, the students incrementally learnt to write code to enable the program to accept user inputs via the mouse and/or the keyboard and to make the objects react to these inputs. They also programmed the sprites to react upon collisions with other sprites. Animation sequences were achieved by way of loops and conditional statements. A sample assignment requirement is illustrated in Figure 2. The final project using Scratch was to create an interactive story using images, sounds, collisions, several sprites, focusing on Why Computer Science matters?

3.3 Programming Assignments in TouchDevelop

The first assignment was an introduction to the interfaces offered by the Windows phone. The student was required to write code to enable the user to input his name and take a picture of him using the front camera and read out a welcome message using the text-to-speech engine of the phone. The students designed a pedometer as the second assignment, to count the number of steps they took using the phone's sensors. They also generated real-time charts of distance they covered against the time they spent. They also used the geo-location interfaces to calculate the distance between two coordinates. The third and final assignment was to come up with an interactive story scene sequence, using the graphic interfaces of the phone. The final project was to design



StoryTitle

The Moral Compass

Start

Z is a 20-something year old man living in the lower-income part of Santa Cruz. Desperate for money,

crime

Eventually, his luck ran out. Z was standing in an alley way, waiting for his client to pick up a

caught

Several years [[later]]... After serving his time in jail, Z was due to be released soon.

later

He was told that he was being considered for the implementation of a [[Moral Compass]]. Z

Moral Compass

"The Moral Compass can alter the way people make decisions by disrupting a certain

hearing

Z arrived in the court room. He looked at the committee who would decide his fate. A panel of four judges

one

<html> <object width="560" height="315"><param name="movie" value="http://www.yout

two

<html> <object width="560" height="315"><param name="movie" value="http://www.yout

three

<html> <object width="560" height="315"><param name="movie" value="http://www.yout

four

<html> <object width="560" height="315"><param name="movie" value="http://www.yout

verdict

When the committee returned to the

The Moral Compass

Z arrived in the court room. He looked at the committee who wo
judgemental eyes, the committee reviewed his criminal record a
his ability to behave morally within society. But he felt as thoug
committee filed out to a room in the back to decide Z's fate.

Each committee member had a strong opinion regarding the pro

Z was released after he recovered from the surgery. At first, everything with the chip app
malfunction. It affected Z's brain in the wrong way. Instead of helping him to make mor
crime spree. He was robbing stores, committing assault, dealing drugs. There was no lim

Rewind
Restart
Share
one
two
three
four
Soon t
Twine tool window with interactive story nodes input by students.
An interactive story compiled in HTML format.

Figure 1: Twine tool window showing nodes comprising the interactive story (left) which is then compiled and represented in a html document (right)

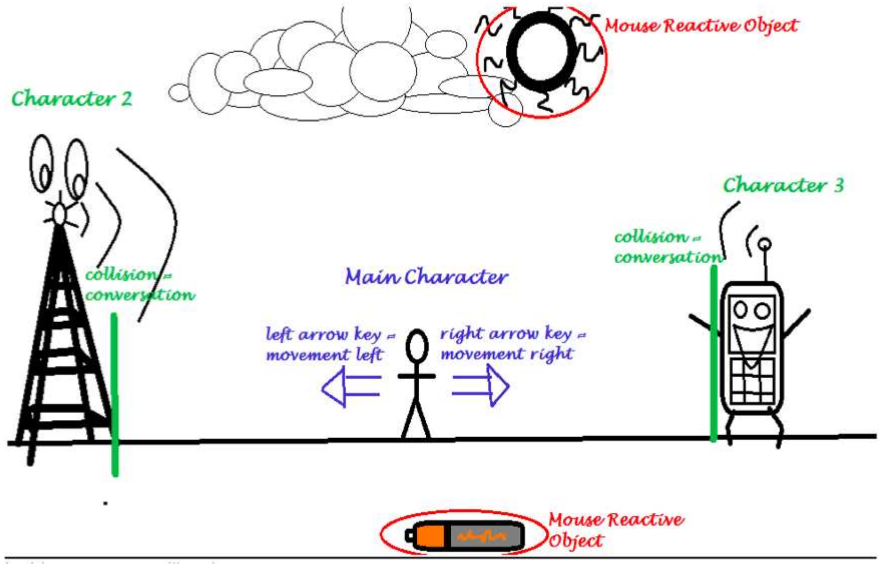


Figure 2: Scratch programming assignment description of sprites and interactivity using mouse and keyboard

a utility or an interactive story on why Computer Science Matters.

Figure 3 illustrates some sample TouchDevelop programming assignment output screens programmed by the students using the Microsoft Windows Phone.

3.4 Flexible Grading Rubric

Students were given an option to choose between Scratch and TouchDevelop. Clearly, TouchDevelop presented a more difficult learning curve and we did not want to require every student to work with TouchDevelop. This decision was guided by the constraint that Microsoft could provide only a small number of Windows Phone for a class consisting of 140 students. Initially 21 students chose to use TouchDevelop and one student switched to Scratch after one week. Points for programming whether in Scratch or in TouchDevelop were the same for all students.

Flexibility in grading rubric was introduced because some students did not appear keen on developing their stories further in Twine while some others did not seem very interested in probing why computer science matters. Yet some others appeared to be very keen in exploring either specific computer science topics such as Turing Machines or questions such as what kind of computer science curriculum is needed for a 21st century learner. In order to motivate the students and channel their talents into topics of their choice, students were given the flexibility to choose between these options: create a powerpoint presentation on why computer science matters, or create an interactive story using Twine, or create a research paper on a computer science topic with prior approval from the instructor.

Students were initially a bit surprised with the flexibility in choosing the task. However, most students welcomed this approach because it allowed them to work on a task of their choice. We feel that it clearly improved the final outcomes. However, reporting of these grades to students became tricky because some of these tasks were graded ahead of others. This created confusion because some students received grades out of 65 while others were receiving grades only out of 55 and did not allow computation of averages and medians consistently across the whole class. In the hindsight, clearly grades should have been reported only after all the work was graded for the whole class irrespective of their chosen option. However, due to constrained resources for such a large class, this approach would have introduced undue delay in reporting timely grades on assignments that could help students to improve their next submissions.

3.5 Course WebSites: eCommons and Piazza

eCommons platform was used to allow students to share ideas so that they can work in groups while working on interactive story telling projects or programming projects. But the platform appeared clumsy and difficult to use. So, we introduced Piazza as an additional website where students could post their interactive stories and view the stories posted by others. In addition, since eCommons is not accessible to the outside world and we wish to disseminate the information on our course, we also used a regular course website. These three websites clearly became overwhelming and confusing because students did not know which site to use for what purpose. This feedback was provided strongly in mid-quarter. Support for Piazza was suspended immediately thereafter and this decision was greeted with cheers.

4. FINAL PROJECTS

Now we describe the outcomes of some final projects in scratch, touchdevelop, powerpoint presentations on why computer science matters, interactive story using twine, and research papers.

4.1 Why CS Matters

One of our major themes while teaching this introductory course to computer science was to expose students to the emerging applications of computer science and to encourage them to appreciate the role played by computer science in defining a better world. Assignments designed to achieve this goal required students to provide concrete examples from the real world where computer science has played a significant role in making a difference to society. The evidence for these examples had to be presented in one of the several ways: a link, an image, a video, sounds, with descriptive text.

Students highlighted the influence of technology on teaching methods by discussing and bringing out examples. They thought it was important how documents of relevance are now accessible and searchable. They cited multiple websites and enterprises which had helped distance education come a long way, eventually making the distance between the instructor and pupil virtually irrelevant. Many examples were brought forth indicating towards the impact of artificial intelligence algorithms on a variety of fields like the stock markets, hospital management, modern surgery, warfare and space exploration.

It was clear that students appreciated the spread of the immense communication network by way of the Internet. They cited websites which were helping source funds as loans and donations from across the world for deprived sections of the society. Assistive technologies that have flourished with the pervasiveness of sensors able to detect location and motion were mentioned as well.

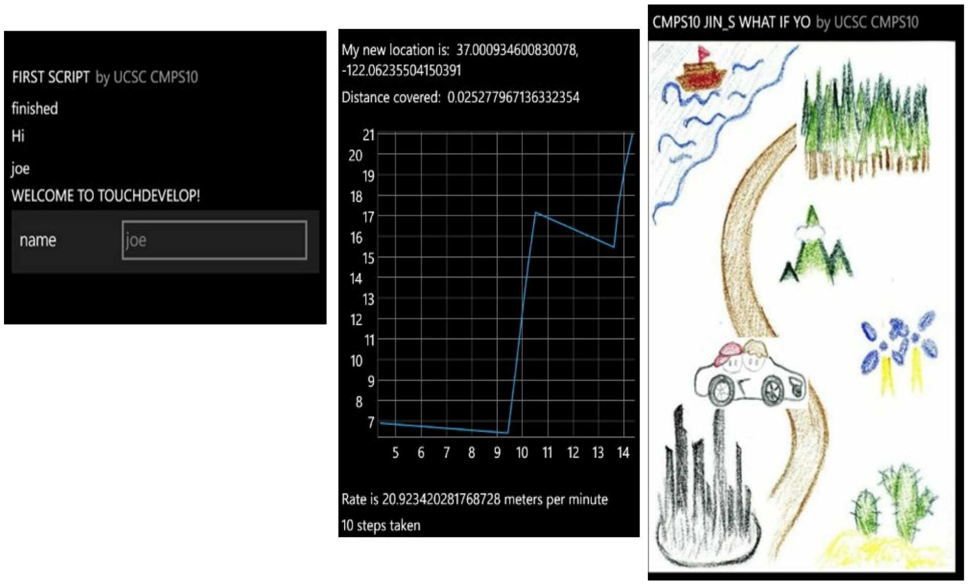
4.2 Research Papers

Majority of students chose to create powerpoint presentations on why computer science matters. Only a handful but dedicated students chose research topics of their interest. Turing machine and computer-assisted proofs were the chosen technically oriented topics. One student discussed the role of computers in solving the Four Color Conjecture.

Topics for research papers included: Online introductory computer science courses, and the needs of a 21st century learner. In a critique of online classes, a student wrote that one-to-one learning that occurs in a tutoring environment cannot be substituted with online classes.

Many students chose to write about the inspiring innovators of today and one chose to write about the computer science awards. Entrepreneurs such as Salman Khan for Khan Academy, Ben Silbermann for Pinterest, Gossy Ukanwokwe for Students' Circle were chosen. One student researched works of award winning computer scientists Luis Von Ahm and Hal Abelson.

Additional topics include open source software, mobile platforms, and the role of computer technology in creating a greener economy. Overall, the quality of research papers was excellent and reflected the commitment of students towards excellence. Many of the students who chose this option received an A or an A+ because their performance in other areas were also equally strong or stronger.


```
FIRST SCRIPT by UCSC CMPS10
finished
Hi
joe
WELCOME TO TOUCHDEVELOP!
name    joe
```

```
My new location is: 37.000934600830078,
-122.06235504150391
Distance covered: 0.025277967136332354
```

```
Rate is 20.923420281768728 meters per minute
10 steps taken
```

CMPS10 JIN_S WHAT IF YO by UCSC CMPS10

Figure 3: Solution screens for the three TouchDevelop programming assignments

4.3 Interactive Story Telling

After the initial assignment on story telling in which the whole class participated, only a small number of students chose to develop their story further in Twine. As in the case of research papers, these students put in tremendous amount of effort and developed compelling stories that presented ethical dilemmas surrounding the use of computer technology to very promising science fiction stories.

We received several excellent storylines that were created with a strong theme - "the influence of computer science on the society". Students didn't limit themselves to the current advancements of the field but built for themselves a burgeoning world of technology, where the science flourished and was more pervasive than ever before. These stories carefully weighed the pros and cons of technology, its usage, its consequences with humanity at the center point. Simultaneously, they presented the vastness and infinite potential that the field had to offer. The outlines of these stories predominantly dealt with the growing intrusiveness of electronic gadgets and software. There were several accounts written, describing the changing security scenarios with upcoming cyber-security challenges. These stories were expressed via Twine and some of them were further articulated using either Scratch or TouchDevelop programs.

4.4 Programming Projects in Scratch

The quality of final projects using Scratch varied as expected. Some of the best projects had very detailed storylines with handcrafted images and sounds made with great care. Most stories brought characters and props to life with handmade images associated with sprites. Almost all the submissions met the technical requirements where students designed interactive Scratch programs involving mouse and keyboard initiated user inputs. Figures 4, 5, 6, and 7 are some screens of the programs created by the students as final projects for the course.

4.5 Programming Projects in TouchDevelop

Figure 4: A story about how a lonely kid learns and later uses computer science to keep himself entertained and informed.

Figures 8, 9, 13, 11, 10, and 12 are some sample phone screen-shots belonging to the final projects demonstrated by students who programmed using the TouchDevelop language. The projects harnessed several interfaces exposed by the phone. In addition to using traditional programming constructs like variables, control flow statements, conditional statements, methods and loops, the students also extensively used the sensors and the phone data to make their programs interactive and useful.

5. SURVEY OUTCOMES AND FEEDBACK

There were 140 students enrolled in the class. Freshmen, sophomore, junior, and senior accounted for 43%, 26%, 17%, and 12% respectively of the class. 68% students were males and 32% students were females. 42% students were asian, 35% were white, 20% were hispanic or latino.

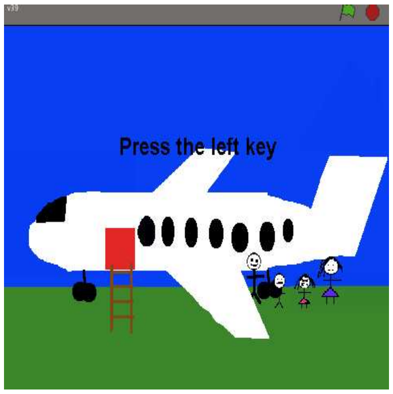


Figure 5: An account of how a family vacation is saved from mishaps thanks to artificial intelligence applications in aviation.

Figure 6: A story highlighting the social importance of online publications like blogs.

Figure 7: An entertaining remake of the movie - "The Matrix"

UCSC INTERACTIVE ALARM by UCSC CMPS10
Enter minute
36
Enter hour
15
UCSC INTERACTIVE ALARM by UCSC CMPS10
Joshua Becker
William Barakatt

Figure 8: An alarm clock that requires the user to find his way through a maze to stop the alarm from ringing.

Figure 9: An audio-visual aid to learn new japanese words using flash cards with english equivalent words, pictures and pronunciation aid.

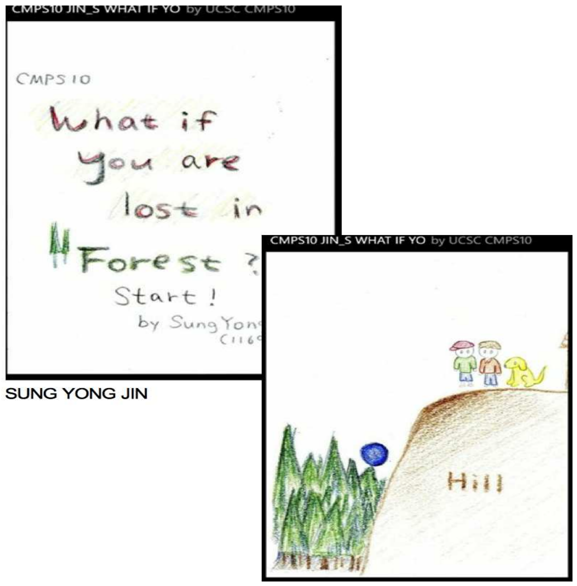

CMPS10 JIN_S WHAT IF YO by UCSC CMPS10
CMPS 10
What if you are lost in Forest?
Start!
by Sung Yon
SUNG YONG JIN
Hill

Figure 10: An interactive story of kids lost in the dark of the forest, and found thanks to geolocation technologies.

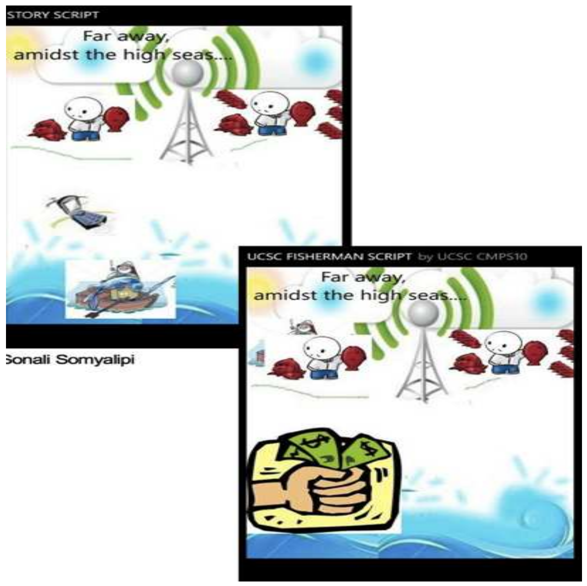

STORY SCRIPT
Far away, amidst the high seas....
Sonali Somyalipi
UCSC FISHERMAN SCRIPT by UCSC CMPS10
Far away, amidst the high seas....

Figure 12: An interactive story highlighting the huge economic change brought about by introduction of mobile phones in the fishing community of south-coastal India

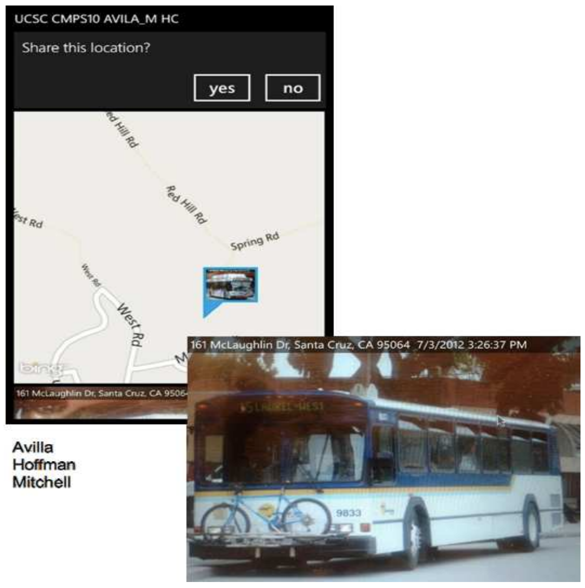


Figure 11: An app that helps school students to track and share real time locations of their school bus.

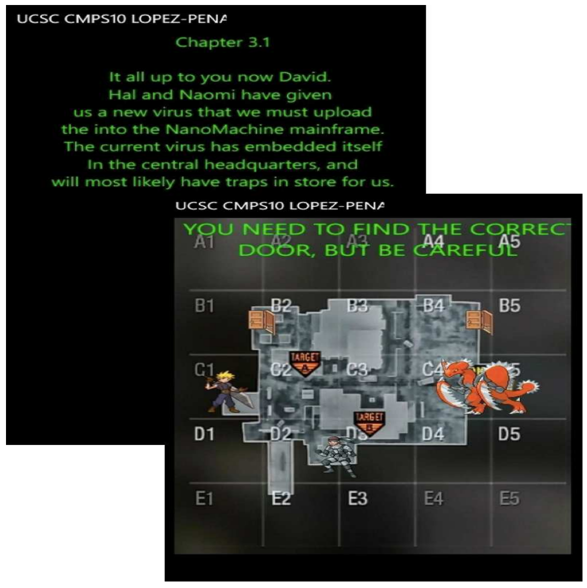

UCSC CMPS10 LOPEZ-PENA
Chapter 3.1
It all up to you now David.
Hal and Naomi have given
us a new virus that we must upload
the into the NanoMachine mainframe.
The current virus has embedded itself
In the central headquarters, and
will most likely have traps in store for us.
UCSC CMPS10 LOPEZ-PENA
YOU NEED TO FIND THE CORREC
DOOR, BUT BE CAREFUL
B1 B2 B3 B4 B5
C1 C2 C3 C4
D1 D2 D4 D5
E1 E2 E3 E4 E5
TARGET

Figure 13: A game where the user needs to steer the sprites to the right doors to win.

30% students belonged to social sciences division, 30% students were undeclared, 14% students were pursuing engineering, 14% sciences, 9% humanities, and 4% arts.

In response to the question on reason for taking the course, 23% students indicated that this was a required general education course, 40% indicated that it was a course required for the major, and 27% took the class because they were interested in the course. The students were administered a pre-course and a post-course survey, both designed along the lines of the AP CS Student survey published by College Board[1]. The pre-course survey helped us know how the students defined computing and what they thought a person skilled in computing could do. The survey asked questions that revealed patterns of technology use by students, and brought out their attitudes towards computer science.There were also questions that helped us know the students' comfort level and familiarity with logic and mathematics, and any previous experiences they had with computers. Lastly we asked questions to ascertain the demographics of the audience to the course.

62% of students had never taken any computer science courses prior to taking this class while 11% had less than one year experience, and 18% students had taken one year of computer science prior to taking the class already. Similarly, about 10% students indicated that they had learned CS on their own for a year, 15% for less than a year, and 67% had not learnt CS on their own prior to taking this class.

87% students use mobile devices for texting and talking in comparison to 48% for social networking and 48% for video or music. Surprisingly, 30% students indicated that they had never played a video game. Also, interestingly, about 40% students stated that they use mobile devices for coursework or creative work at least once a week. This provides an opportunity to draw these students into TouchDevelop.

The post-course survey again asked how the students defined computing and what they thought a person skilled in computing could do. Next, their attitudes towards computer science were gauged again, with questions concerning their comfort and interest in learning computing concepts. They then self rated their abilities in using a computer in their day-to-day activities, solving logic problems, writing computer programs and solving problems with computer science, while effectively analyzing the social implications of computing.

In addition to these pre- and post-course survey, university-sponsored evaluation of the course and the instructor were recorded anonymously. Further a focused questionnaire was administered to students who chose TouchDevelop.

Results

Prior to the course most students said that learning computer science would help them to fix their personal computers, help them build websites and get them a job. Many indicated that they would be able to write programs. After the course, many of these reasons remained, but new reasons were added more prominently. Students expressed confidence in continuing programming with TouchDevelop and Scratch and also stated they expect to learn programming with other languages and start using other software packages. By the end of the course, 84% of the students said they could now see, how they could apply their knowledge of computer science to solve problems of human importance.

In response to the question, "if someone were to ask you things using or requiring "computing", which things would you be thinking about", the responses in percentages were as follows: (i) using computer applications like Word, GIMP, Blender, Photoshop: 79%, (ii) developing a webpage: 70%, (iii) using online social media: 60%, (iv) searching for data for a research paper on the web:56%, (v) searching for answer to some non-school related question:56%, (vi) fixing computer problems:64%, and (vii) solving a problem of importance to humans using computer science:84%.

In response to the question "I had a strong desire to take this course", 37% students somewhat or strongly agreed, 42% were neutral, and 21% students somewhat or strongly disagreed. The results of "I gained a good understanding of the course content" is strikingly similar: 46% reported strongly or somewhat agreed, 37% remained neutral, and 18% somewhat or strongly agreed.

Although the overall rating of the course as a learning experience decreased slightly from a prior rendering of the course by the same instructor using C++, 33% of students indicated that they are more likely to take additional computer science courses as compared to 21% who are less likely.

Students' Feedback on Scratch and TouchDevelop We also administered a survey to get student feedback on the languages and platforms used in the class. The students answered questions regarding their experiences using TouchDevelop on the Microsoft Windows phones/Scratch Programming. The students who chose TouchDevelop Programming said that they had a richer programming experience with phone sensors, and structured programming constructs. Using TouchDevelop programming they could create phone applications that were practically useful. However, learning to use the limited phone interface to input code involved a slightly longer learning curve.

On the other hand, students who learnt programming with Scratch learnt basic programming constructs as well as could use rich multi-media content like graphic images and sound clips in the programs they wrote. The Scratch Programming language provided to them an intermediate platform for expressive story telling and game design. The learning curve for Scratch was acknowledged to be much shorter than that of TouchDevelop.

6. DISCUSSION AND FUTURE WORK

Students greeted an earlier offering of the introductory CS class with little programming language at UCSC with very high ratings. Since these type of courses have been successful at other institutions as well, we need to firm up this approach and identify components – topics, use of guest lectures, use of powerpoint presentations – that can create a meaningful learning experience for students. It will be useful to determine how well they were engaged with or motivated by the class. A well designed pre- and post-survey can begin to capture these elements.

The tailored courses such as introduction to computer graphics or animation using high level software packages such as Maya have been also tremendously successful. A newly introduced course on social networks also seems to be popular. It is worth contemplating introduction of other tailored lower division computer science courses such as computational biology, computational finance, computational photography, computational media, or computational gadgets. An interesting proposition, given the emphasis on social causes at our campus, would be a course on computational social science (with focus on social issues). Experience at other institutions suggest that these tailored courses are likely to be well received by students.

This brings us back to our key question: what about general-purpose introductory computer science course with programming component? Students have been giving acceptable but not outstanding evaluations to the traditional offering of the course using C++. Nevertheless, as stated before, focused discussions with students suggest that this may not be the ideal way to offer the class with its traditional focus on searching, sorting, algorithms without much introduction or exposure to current gadgets and media. Our recent experiments with using *Scratch* seem to have been very successful, although the data is limited only to a summer session, and therefore, not conclusive. We also need to address the question whether scratch is powerful enough to convey the depth of computer science concepts. On the other hand, initial experimentation with the use of *processing* or *TouchDevelop* have received lower overall learning experience scores from students. However, course evaluation results do not separate the experience based on programming language from other components of the course. It appears that other aspects of the course such as flexible grading scheme, multitude of course web sites, and attempt to cover additional material such as Twine etc. could have been overwhelming. Focused informal conversation with TouchDevelop students indicated that many of them were very satisfied and in fact proud of their final projects. Similarly, final projects in Scratch during Summer 2011 and during Spring 2012 also produced several very strong projects. Other institutions are also using Python to teach computer science for non-majors and have reported success informally. More precise or component-wise evaluation of course will be needed to assess which programming language or which aspects of the course are most compelling and interesting to the students.

In summary, the key goal of introductory computer science courses should be to engage and motivate students. Non-programming introduction to computer science and tailored computer science courses appear to be doing a better job of achieving this goal. Our course succeeded in communicating the importance of computer science in solving problems of importance to society. Further research is needed to determine which programming languages or combinations should or could be used for introductory computer science course.

7. ACKNOWLEDGMENTS

We would like to thank the undergraduate tutoring staff-Carmelo Herrera, Christopher Taylor, Steven Butkus, Michael Okuma, Wyatt Sanders, Brice Gatelet, and Zane Mariano for their dedication, knowledge and creative exchange of ideas on curriculum design and delivery. We also express our gratitude to Siddhant Chandgothia for assisting us with TouchDevelop applications and presentation. We would like to thank Jim Whitehead, Marilyn Walker, Charlie McDowell for helpful discussions on redesigning the curriculum for the introductory computer science class at UCSC. We are also grateful to James Davis, Christopher Yonge, Jim Whitehead and Linda Werner for guest lectures. We also wish to acknowledge our gratitude and appreciation to Microsoft researchers- Nikolai Tillman, Peli de Halleux, Arjmand Samuel, and Kent Foster for sharing their knowledge on TouchDevelop and to Microsoft for their generous support. Most of all we are very thankful to the students of CMPS10 Spring 2012 class for their contribution and creativity in their final projects on Twine, Scratch, and TouchDevelop which made this teaching effort extremely worthwhile. In particular, we want to acknowledge the final projects on TouchDevelop by Joshua Becker, William Barakatt, Christian Lopez Pena, Christopher Mimilis, Nicholas Dahl, Sung Yong Jin, Mario Avilla, Charles Hoffman, Alexander Mitchell, Miles Kropf and Maxwilliam Chao; final projects in Scratch by Jason Alvarez, Johnson La, Keesha Bush-Trennery, Andrew Goldsby, Abraham Gonzales and Seok Won Kim; final projects on Twine - Jaspreet Dayal, Anthony Moreno, Lawrence Saguilig, Alexander Tanner; final projects on Why CS Matters by Shadie Nimri, Gina Kim, and Craig Collins.

8. REFERENCES

[1] College board. http://about.collegeboard.org/.

[2] Partnership for 21st century skills. http://www.p21.org, 2011.

[3] M. Ben-Ari. Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(1):257–261, 1998.

[4] G. Carmichael. Adding computer science to an introductory computing class for non-majors. http://gailcarmichael.com/sites/default/files/, sigcse2011paper.pdf, 2011.

[5] S. Cooper and S. Cunningham. Teaching computer science in context. *ACM Inroads*, 1(1):5–8, Mar. 2010.

[6] T. J. Cortina. An introduction to computer science for non-majors using principles of computation. In *Proceedings of the 38th ACM technical symposium on Computer Science Education*, SIGCSE '07, pages 218–222, New York, NY, USA, 2007. ACM.

[7] N. R. Council. *Report of a Workshop on the Scope and Nature of Computational Thinking.* National Academies Press, 2010.

[8] N. R. Council. *Report of a Workshop of Pedagogical Aspects of Computational Thinking.* National Academies Press, 2011.

[9] A. Forte and M. Guzdial. Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *Education, IEEE Transactions on*, 48(2):248–253, 2005.

[10] S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking. A multidisciplinary approach towards computational thinking for science majors. *SIGCSE Bull.*, 41(1):183–187, Mar. 2009.

[11] D. Joyce. The computer as a problem-solving tool: a unifying view for a non-majors computing course. In *Proceedings of the 29th ACM technical symposium on Computer Science Education*, SIGCSE '98, pages 63–67, New York, NY, USA, 1998. ACM.

[12] C. Klimas. Twine. http://gimcrackd.com/etc/src/.

[13] L. Lee. A non-programming introduction to computer science via nlp, ir, and ai. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1*, ETMTNLP '02, pages 33–38, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[14] R. Libeskind-Hadas. Computational thinking courses for non-majors. http://www.cs.hmc.edu/ hadas/IntroCourses.html, 2012.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Commun. ACM*, 52(11):60–67, Nov. 2009.

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

[17] R. Stross. Computer science for the rest of us. http://www.nytimes.com/2012/04/01/business/computer-science-for-non-majors-takes-many-forms.html, Mar. 2012.

[18] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich. Touchdevelop: programming cloud-connected mobile devices via touchscreen. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, ONWARD '11, pages 49–60, New York, NY, USA, 2011. ACM.

[19] N. Tillmann, M. Moskal, J. de Halleux, M. Fähndrich, and T. Xie. Engage your students by teaching programming using only mobile devices with touchdevelop (abstract only). In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 663–663, New York, NY, USA, 2012. ACM.

[20] I. Utting, S. Cooper, M. Kölling, J. Maloney, and M. Resnick. Alice, greenfoot, and scratch – a discussion. *Trans. Comput. Educ.*, 10(4):17:1–17:11, Nov. 2010.

[21] B. Ward, D. Marghitu, T. Bell, and L. Lambert. Teaching computer science concepts in scratch and alice. *J. Comput. Sci. Coll.*, 26(2):173–180, Dec. 2010.

[22] J. Wing. Research notebook: Computational thinking – what and why? http://link.cs.cmu.edu/article.php?a=600, June 2011.

[23] L. E. Winslow. Programming pedagogy – a psychological overview. *SIGCSE Bull.*, 28(3):17–22, Sept. 1996.