Teaching Introductory Programming Concepts: A Comparison of Scratch and Arduino

Anne Beug, Phillip L. Nico
Department of Computer Science,
California Polytechnic State University, San Luis Obispo

Abstract

In this paper we present our experiences developing and delivering two separate introductory computer programming units for high school students—one based on the Scratch visual programming environment and the other based on the Arduino embedded system prototyping platform. Scratch is a well-proven educational software development platform that teaches core programming concepts through a graphical programming interface, aimed at junior high and high school-aged students. The Arduino platform consists of both hardware and software: an open source microcontroller system programmed in a C-like language. We developed parallel curricula in Scratch and Arduino and compared the two in the setting of five high school classrooms. Each course consisted of five sessions (with a lecture and a lab), each covering a different topic, building on previous sessions. While the results of our quantitative study have not been conclusive, our experience suggests that the Arduino platform is not yet ready for teaching core programming concepts to computing novices. The combination of the C-like language and the hardware were too complex for novice programmers to use in learning programming concepts.

Introduction

We performed a study to evaluate the suitability of the Arduino platform in teaching core computing concepts to high school students. We held series of five sessions with various classes at two local high schools—both programming classes and computer application (Word, Excel, etc.) classes. Students in the classes had diverse educational and computing backgrounds—some had no computing education and did not feel comfortable with computers and others had completed an AP Computer Science course in Java.

During each class session, we covered a core programming concept, with each session building on previous sessions. The first session introduced the students to computer programming, as well as either the Scratch or Arduino programming environment. During the remaining sessions, we introduced the concepts of variables, conditionals (if-else statements), iteration (loops) and functions. We wanted to see how well each environment would work for teaching each concept and programming in general.

We assessed the students' grasp of the chosen concepts and experiences through a pre-survey and a post-survey with quantitative and qualitative questions. Both surveys also asked the students about their computing background and attitudes toward computing.

Platform Choice

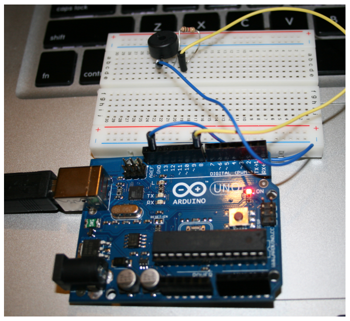
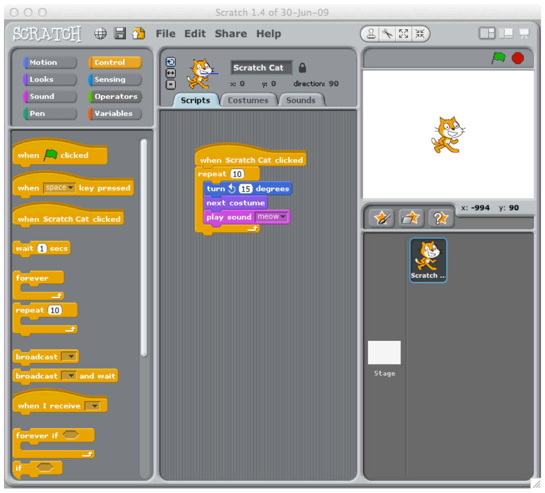


Figure 1: Arduino microcontroller

We chose Arduino as our experimental platform to study because of its growing popularity with electronics hobbyists [14] and recent introduction into embedded systems curricula [11] [14]. We found a dearth of research on using Arduino to teach introductory programming education. The Arduino platform consists of a set of microcontrollers, a programming language and an IDE. All components of the platform are open source. The language is based on the Wiring and Processing [1] languages that were created to teach core programming and computing concepts through electronics and visual arts respectively. Arduino as a language is syntactically similar to C and Java.



```
when Scratch Cat clicked
repeat 10
    turn 15 degrees
    next costume
    play sound meow
```

Figure 1: Scratch development environment

Scratch, on the other hand, grew out of academic work in MIT's Lifelong Kindergarten Lab, officially launching in 2007 as a new educational programming and computing platform. From its website, "Scratch is designed to help young people (ages 8 and up) develop 21st century learning skills. As they create Scratch projects, young people learn important mathematical and computational ideas, while also gaining a deeper understanding of the process of design. With Scratch, kids can create their own interactive stories, games, music, and animations… [2]"

Scratch's visual programming interface allows users to build programs by selecting "blocks" (programming instructions) from a palette on to a script area. The blocks click together only in meaningful ways, preventing syntactical errors.

Because Scratch does not have built in functions, we taught the last Scratch lab (functions) in BYOB 3.1 (Build Your Own Blocks). BYOB is an extension to Scratch that includes the ability to create custom blocks [13].

Multiple studies have been performed investigating the effectiveness of Scratch in teaching introductory programming concepts [6][7][8][15][16][17]. It works well for some concepts (loops and conditionals) but not for others (variables and functions) [8]. Scratch has been accepted as a platform for teaching novice programmers in junior high [5], high school and at the university level [7][8][15][16][17][18].

The other platform we considered for our control group was Alice [19]. However, we chose to use Scratch as it is already used in many of our university outreach programs as well as in the local elementary and secondary school district. Using BYOB in the final lab alleviated our main concern with Scratch–that it does not have the capability to express functions.

Related Works

There is a diverse and large body of research in the area of computer science education, focused on primary, secondary and tertiary schools, starting in the 1970's [3]. One common theme in many papers is the fact that learning and teaching programming is difficult [3][4][5]. Other work studies the specific problems novice programmers encounter. Pea writes about different "conceptual bugs" in novice programming [4] – misunderstanding the order in which programs execute, attributing intentionality to programs and assuming the programs can read the programmer's mind.

Studies have found various results using Scratch to teach core computing concepts. Colleen Lewis reports students learned conditional statements better through Scratch than Logo [6], but, surprisingly, did not show greater comprehension of loops. In "Habits of Programming in Scratch," Meerbaum-Salant, et al. [7] found that while Scratch encourages self-directed learning, students only really learned programming concepts when explicitly taught the concepts. Rivzi, et al. [8] describe a new Scratch-based undergraduate course (CS0) inserted before the traditional first programming course (CS1) aimed to increase student retention. One set of students enrolled directly in CS1, while the others enrolled first in CS0. Amongst the CS0 set, the researchers found increased interest in computer science as well as improved learning outcomes. Another study investigated learning results of computer science concepts by students learning in the Scratch environment [9]. A Scratch-based curriculum was developed for middle-school-aged children. Middle school teachers taught the course during regular school hours. Students performed well with loop concepts, but less so with variables.

Far less research has been performed with Arduino, perhaps because it was not conceived as an educational platform. Most work describes integrating Arduino in to existing microcontroller or robotics courses [10][11][12].

Courses

We taught the parallel courses at two high schools to a total of five classes. Two of the classes were computer applications classes—Word, Excel, PowerPoint. Both of these groups, of which few students had any programming experience, completed the Scratch version of the course.

Another two classes were programming courses—a mix of first, second and third semester programming students. These groups completed the Arduino version of the course. The final class was a manufacturing concepts class that included a section on electronics. This group completed sections 1 – 4 of both the Scratch and Arduino versions of the course.

The course itself was divided in to five sections – introduction, variables, conditionals, iteration and functions. Each section included a short 10-minute lecture introducing the given concept through analogy and examples. The rest of the time (between 40 – 70 minutes, depending on the school, course and day) was spent with the students working on a lab exercise either individually or in small groups, depending on available resources.

The Scratch and Arduino labs are described in Table 1.

Lab	**Scratch**	**Arduino**
1 – Introduction	Create animation with multiple sprites.	Blink LED off and on.
2 - Variables	Create MadLibs-style word game, using variables to store user-entered data.	Use potentiometer to create dimmer light switch.
3 – Conditionals	Create animated tag game with user controls based on keyboard input.	Create push button light switch to control an LED.
4 – Iteration	Create interactive musical animation with multiple sounds.	Play a song using a Piezo buzzer.
5 – Functions	Create calculator with sum and average functions.	Read temperature (in °C) and write function to convert to °F.

Table 1: Labs

For the Scratch labs, students built game-like programs. In each, they were able to either draw their own sprites (graphical programmatic elements) or use Scratch-provided sprites. Many students chose to spend time drawing their own sprites. In the fourth lab, we explicitly asked students to include sound in to their program; however, most had already been using sound starting with the first program. Students also discovered that they could download others' Scratch programs from the Scratch website and incorporate parts of those programs into their own or extend those programs.

The Arduino labs required much more work to set up. Before each session, we had to wire up the boards, LEDs and other electronic components. One significant challenge these classes faced was getting their lab computers to communicate with the Arduino boards. It took us the first couple sessions to iron out all the problems with the Arduino drivers. And once all the boards were set up correctly, it was easy for students to jostle the boards enough to loosen wires. Once wires cam undone, students did not have enough electronics background to read the provided wiring diagrams.

Survey

Feedback on the course was gathered through a short (10 -15-minute) pre- and post-survey. The pre-survey contained a subset of questions from the post-survey. Of the five groups, three groups reported an increase in "comfort with computing," while two groups (one working with Scratch the other with both Scratch and Arduino) related a decrease in comfort.

In all groups, there were a total of 119 participants. Of those participants, 93 completed the pre-survey (78.15%) and 85 (71.43%) the post-survey; 59 (49.58%) completed both the pre- and post-surveys, 34 (28.57%) only the pre-survey and 26 (21.85%) only the post-survey.

Overall, we question the accuracy of the survey answers–for instance, in one case, a student lost an entire year of programming experience between taking the pre-survey and the post-survey, six weeks apart. This points out the possibility that students were confused by the survey questions. In addition, students were not given any class credit for participation, program or survey completion.

The quantitative objective learning assessments came back with mixed results for both Scratch and Arduino. Some students showed an improvement in answering the questions correctly after going through the course; however, other student which had initially answered the questions correctly in the pre-survey, failed to answer them in the post-survey.

At the end of the following code, what is `c` equal to?

```
a = 3
b = a - 1
a = b * 2                c is __________
c = a + b + 1
```

Table 2: Sample quantitative survey question

From the students free-form comments on the course, the most common things students liked was that the course was "fun" and "interesting." The dislikes included that the course was "too hard," "boring" and "confusing". Students from the Scratch courses frequently mentioned enjoying drawing their own sprites and being able to add sounds into their programs. On the other hand, students in the Arduino courses expressed frustration at getting their Arduino boards to work at all with their computers.

A full listing of survey questions and responses can be found in [13].

Future Work

We found two interesting areas of future work—first, to further the study in a controlled environment such that the outcomes for the two groups (Scratch learners and Arduino learners) could be directly compared; and next, to study how students translate learnings from the two courses into further course work—either AP Computer Science in Java or CS1 courses in a university.

Conclusion

Based on our observances in the two parallel courses, we conclude that Arduino is not a suitable platform for teaching introductory programming to high school students. The platform overwhelmed the novice students with the addition of the hardware element to the software element. Scratch was much easier for the new programmers to pick up quickly, with the exception of the fifth lab (functions in BYOB), which proved to be overly complex.

Both high schools, though, will continue to integrate more computing in to their classes. They intend to use Scratch in their computer applications classes and Arduino in their programming classes, after an introduction through Scratch.

References

1. "Overview \ Processing.org." [Online]. Available: http://processing.org/about/. [Accessed: 09-Apr-2012].
2. "Scratch Homepage." [Online]. Available: http://scratch.mit.edu/. [Accessed: 09-Apr-2012].

A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," Computer Science Education, vol. 13, no. 2, pp. 137–172, 2003.

3. L. McIver and D. Conway, "Seven deadly sins of introductory programming language design," in Software Engineering: Education and Practice, 1996. Proceedings. International Conference, 1996, pp. 309 –316.
4. R. D. Pea, "Language-independent conceptual' bugs' in novice programming," Journal of Educational Computing Research, vol. 2, no. 1, pp. 25–36, 1986.
5. C. M. Lewis, "How programming environment shapes perception, learning and goals: logo vs. scratch," in Proceedings of the 41st ACM technical symposium on Computer science education, New York, NY, USA, 2010, pp. 346–350.
6. O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," in Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, 2011, pp. 168–172.
7. M. Rizvi, T. Humphries, D. Major, M. Jones, and H. Lauzun, "A CS0 course using Scratch," J. Comput. Sci. Coll., vol. 26, no. 3, pp. 19–27, Jan. 2011.
8. O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning computer science concepts with scratch," in Proceedings of the Sixth international workshop on Computing education research, 2010, pp. 69–76.
9. P. Bender and K. Kussmann, "Arduino based projects in the computer science capstone course," J. Comput. Sci. Coll., vol. 27, no. 5, pp. 152–157, May 2012.
10. P. Jamieson, "Arduino for Teaching Embedded Systems. Are Computer Scientists and Engineering Educators Missing the Boat?," in International Conference on Frontiers in Education: Computer Science and Computer Engineering, 2011.
11. R. Balogh, "Educational robotic platform based on arduino," in Proceedings of the 1st international conference on Robotics in Education, RiE2010. FEI STU, Slovakia, 2010, pp. 119–122.
12. "Build Your Own Blocks Homepage." [Online]. Available: http:// http://byob.berkeley.edu/. [Accessed: 05-Jan-2013].
13. A. Beug, "Teaching Introductory Programming Concepts: A Comparison of Scratch and Arduino." [Online]. Available [Accessed: 03-10-2013].
14. J. Sarik and I. Kymissis, "Lab kits using the Arduino prototyping platform," in *Frontiers in Education Conference (FIE), 2010 IEEE*, 2010, p. T3C–1 –T3C–5.
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*, vol. 52, no. 11, pp. 60–67, Nov. 2009.
16. D. J. Malan and H. H. Leitner, "Scratch for budding computer scientists," in *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, New York, NY, USA, 2007, pp. 223–227.
17. U. Wolz, H. H. Leitner, D. J. Malan, and J. Maloney, "Starting with scratch in CS1," *ACM SIGCSE Bulletin*, vol. 41, no. 1, pp. 2–3, 2009.

18. U. Wolz, J. Maloney, and S. M. Pulimood, " 'Scratch' your way to introductory CS," *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 298–299, 2008.
19. "Alice.org," *Alice.org*. [Online]. Available: http://alice.org/. [Accessed: 28-May-2012].