1-1-2011

The Effects of Tinkerability on Novice Programming Skill Acquisition

Tian Luo
Old Dominion University, tluo@odu.edu

Original Publication Citation

Luo, T. (2011). *The effects of tinkerability on novice programming skill acquisition.* Paper presented at the World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education.

The Effects of Tinkerability on Novice Programming Skill Acquisition

Tian Luo
Instructional Technology
Ohio University
United States
tl303308@ohio.edu

Abstract: This paper reports on an exploratory study which used a graphical programming environment, Scratch, to facilitate the comprehension of a scripting programming language, ActionScript. Online survey questionnaires were distributed to 34 enrolled students, in a graduate level programming course with a 70% response rate. Findings indicated that Scratch contributed to the understanding of basic programming concepts such as event handling, sequential, and conditional statement but it was less helpful in assisting students' understanding of more abstract concepts such as variables. This study also suggests that students' learning style preference and proficiency with programming also make a difference in their perception of using Scratch. Educators are recommended to provide students with explicit guidance on how the different programming environments manifest similar programming ideas and concepts.

Introduction

Why is it Difficult to Learn Programming?

Programming has been often perceived as one of the most difficult skills to master among all the modern subjects (Lahtinen, Mutka, & Järvinen, 2005). Research studies indicate that its difficulty resides not only in understanding abstract concepts but in designing or generating a functional program to solve tasks based on this understanding (Brusilovsky, Kouchnirenko, Miller, &Tomek, 1994; Robins, Rountree, & Rountree, 2003).

Researchers have investigated programming education and suggested that different strategies should be available for senior, junior, and novice programmers (Robins, et al., 2003) due to different characteristics between expert and novice programmers. Expert programmers tend to be able to proficiently associate their knowledge schemas with problem-solving strategies such as debugging (Linn & Dalbey, 1989), while novice programmers lack the strengths that experts have and often fail in applying their acquired knowledge into designing a program. They often learn programming line by line without being able to comprehend the overall program structure, and constantly find it difficult to combine and arrange the codes into functioning programs (Winslow, 1996).

Why We Learn Programming?

Although programming languages are one of the most challenging subjects to master, learning programming has gained increasing attention to date. In the 21st century, when most of the younger people are considered *digital natives* (Prensky, 2001), programming is deemed as a critical skill to acquire (Rushkoff, 2010). As Resnick, Kafai, & Maeda (2003) stated, being able to program greatly expands the range of what the users can create and how they can express themselves with the computer.

Furthermore, programming promotes *computational thinking*, which involves analyzing and organizing data and other problem-solving strategies that can be applied to non-programming domains (Wing, 2006). Programming involves the creation of "external representations of problem-solving processes" (Resnick et al., 2009, p.3) and it also provides one with opportunities to ponder users' own thinking, or even to "think about thinking itself" (diSessa, 2000). In many cases, the motivation behind learning programming and understanding programming concepts does not end with the mastery of programming ability itself; instead, it fosters a creative, systematic way of thinking, which is greatly needed in this day and age (Resnick et al., 2009).

Introduction to Scratch

What is Scratch?

Scratch is a visual programming environment which was created by the Lifelong Kindergarten Group from the MIT Media Laboratory in partnership with Yasmin Kafai's group at UCLA (Maloney, Peppler,Kafai, Resnick, & Rusk, 2008). It is a programming language designed specifically for novice programmers, which supports programming activities that interest young programmers. Scratch enables them to create interactive, media-rich projects such as animated stories, online news shows, games, book reports, music videos, science projects, simulations, tutorials, music projects and interactive presentations. Scratch also encourages them to share their projects with one another on the online community.

A Scratch project has multiple components, which essentially encompasses a stage (also called a background), and a series of movable objects called sprites. Programmers can import built-in sounds and images to a sprite and enable the mobility of a sprite by using variables and scripts. Meanwhile, they can also create their own images and record their own sounds through the paint tool and sound recorder.

Figure 1: Screenshot of Scratch Interface

As demonstrated in Figure 1, Scratch interface is separated into four sections. On the right hand side is the stage. Programmers can maximize the stage to a full screen mode to showcase a completed project by clicking the button on the bar below the stage. The section below the stage displays thumbnails of all sprites in the Scratch project. One can select a sprite by simply clicking on its corresponding thumbnail. The middle section displays all the scripts executed in the Scratch project. The left-most section is the palette, which is divided into eight color-coded categories of command blocks that can be dragged into the scripting section. These four sections and the top drop-down menu constitute the overall interface of Scratch.

What can Scratch do for Programming?

Novice programmers can program in Scratch by dragging command blocks from a palette into the scripting pane and then assemble them to create various stacks of blocks (Maloney, 2008). Various blocks can be added on top of a stack of blocks to trigger that stack to be responsive to certain run-time events, such as mouse clicking or keyboard events.

In addition, Scratch provides a number of features within its interface. First, Scratch employs a single-window user interface to make navigation uncomplicated. Scratch is always live so there is no distinction in compilation step or edit/run mode. People can remain *on-task* in testing, debugging, and improving their projects without understanding the mechanisms of compilation. Second, the interface allows for user experimentation "with commands and code snippets the way one might tinker with mechanical or electronic components", which is called *tinkerability* by its developers (Maloney et al., 2008, p.17). The feature, tinkerability, enhances people's hands-on learning, embraces a bottom-up approach of syntax writing and facilitates user comprehension of the functionality of blocks. An execution function is made visible so that users can receive immediate feedback, command sequence and flow of control. Tinkerability is a key feature in Scratch as it reinforces the iterative, incremental programming design process that is critical to novice programmers.

Another feature is that no error messages are shown to users so they are free of handling tedious errors. In addition, a variable can be placed on the stage, which monitors activity in order to assist users in understanding the effects of each command. Therefore, novice programmers only need to know a small set of commands to build whole projects (Maloney, 2010).

What Concepts does Scratch Teach?

The programming concepts are taught through manipulating the command blocks in the Palette and putting them into the Scripts section. As the command blocks are categorized into eight groups, respectively, motion, looks, sound, pen, control, sensing, operators, and variables, the corresponding programming concepts are taught by changing the motion, look, sound, or other functionalities of a sprite. For example, under the Motion category, programmers are to set Cartesian coordinates to change its position and mobility. Similarly, programmers can also manipulate numbers in the Looks, Sound, and Pen categories and thus alter its image transformations, musical notes, and so on. Currently, Scratch accommodates arithmetic, comparison and simple Boolean operations. Later it will be able to support higher-level scientific functions such as sine will (Maloney et al., 2008). Through using these commands to manipulate numbers and see the corresponding change on a sprite, young programmers will have a better understanding of numbers as parameters in a programming language.

Scratch also teaches multiple control structures under the Control category. For example, *forever* and *repeat* can be used for understanding of looping (repeating a series of instructions) in programming language. The command blocks, *when key pressed* and *when sprite clicked* demonstrates event handling concept – programmers will see how the sprite responds to events triggered by the user or another part of the program. Synchronization is also enabled in Scratch by allowing a set of commands performing collaboratively— one wait for all triggered scripts to complete before broadcasting.

More advanced programming concepts are introduced in the Operators and Variables categories. *and, or, not* are examples of Boolean logic which reside in the Operators category. Programmers can also create their own variables using commands under the Variables category. Two different kinds of variables, sprite variables and global variables, are supported by Scratch (Maloney et al., 2008). It is intuitive to understand that sprite variables are visible to the scripts limited to the specific sprite, while global variables are visible to all objects in Scratch. Global variables are often harnessed in order to pass data between objects.

Significance of the Research

As the difficulties in learning programming language were discussed previously, Scratch was recently developed as a new programming language learning environment that encourages the novice learners to better understand programming concepts. Although previous research suggested that it makes learning programming more accessible, meaningful and more tinkerable (Maloney et al., 2008; Maloney et al., 2010), there are few exploratory research studies conducted elsewhere to investigate Scratch's impact on learning other programming languages. Furthermore, although the developers claimed Scratch is *a programming for all* (Maloney et al., 2010), there is only

anecdotal evidence to support the statement. It is also unclear whether Scratch's paradigm is useful for a wide variety of users with various experiences, ages and other backgrounds.

Research Questions

This study investigates the following questions:
1. Does a Scratch, the graphical programming environment, help users learn to program?
2. What are the programming concepts that students learn most from Scratch?
3. Do learners use the online learning community specifically designed for potential users?
4. Does the use of Scratch ease the transition to a more advanced scripting language?
5. Do novice programmers have similar experiences as intermediately skilled programmers?

Method

The design methodology used in this research study was a questionnaire survey. The first part aimed to examine learners' engagement in Scratch and its potential relationship with their familiarity with programming language. The second part was meant to pinpoint the specific concepts that students learn most from Scratch. The third part dealt with the overall perception of Scratch and the Scratch community. The last part was the demographic questionnaire. Two open-ended questions inquired learners to describe their experience. At the end of the quarter, email surveys were distributed to the 34 students in the course. The response rate was 70%.

The participants in this research were students from an introductory programming class for educators. The primary programming language learned in this course was Actionscript 3.0, a command line language based upon the ECMAScript specification. The participants were all graduate level education majors. Scratch was introduced in the first week of the class as an initial exposure to a programming environment. Since Scratch is relatively self-explanatory and intuitive, students were asked to finish a project using Scratch after two weeks. Students were not given any guidance or assistance from the instructor but the materials from the online Scratch community. For the project, students were asked to use all eight Scratch categories, which are eight different types of basic programming concepts, and to write a description of this project. Students were also encouraged to explore the environments further after the required completion of their first project.

Results and Analysis

Learner's Profile

The first three questions asked students to inquire about students' familiarity with programming language. Among those students who have programmed, three of them have had more than ten years of programming experience; four of them have had more than five years. According to Winslow (1996), it takes roughly ten years to turn a novice into an expert programmer. Therefore, some of students can be classified as expert programmers, or at least senior programmers.

Understanding of Programming Concepts

As seen from Table 1, Event handling (i.e. when key pressed, when sprite clicked), Conditional statements (if, if-then), and Sequence (think of the order of steps, sequential programming) are the top three conceptual ideas that learners reported having improved in the most from playing with Scratch. These three concepts were followed by Object-oriented programming (consider sprite as an object) and Keyboard input (ask and wait for users to type in the text-box). In contrast, Variables and User interface design are the two concepts which learners reported that they valued least in regards to Scratch.

In the open-ended questions section, some of the respondents also mentioned the concepts that they learned most from Scratch, which are in accordance with the survey results. One of them stated, "It helped me with conditional statements the most. The colorful blocks in Scratch were helpful because I consider myself to be a visual learner. Scratch gives me a better picture than ActionScript." Another student commented that it is a preferred

#	Concepts	SD	D	SD	SA	A	SA	Mean	SD
6	Event handling (i.e. when key pressed, when sprite clicked)	0	2	3	3	10	6	4.63	1.24
1	Conditional statements (if, if…, then...)	1	1	2	4	12	4	4.54	1.25
2	Sequence(think of the order of steps, sequential programming)	0	4	3	4	8	5	4.29	1.40
12	Object-oriented programming (consider sprite as an object)	0	3	2	9	7	3	4.21	1.18
7	Keyboard input (ask and wait for users to type in the text-box)	1	2	4	4	10	3	4.21	1.35
4	Iteration (looping, forever and repeat)	1	3	3	6	8	3	4.08	1.38
8	Random numbers (i.e.pick random integers within a range)	2	1	5	4	9	3	4.08	1.44
9	Boolean logic(i.e and,or, not)	1	4	3	3	10	3	4.08	1.47
11	Coordination and synchronization (i.e. broadcast, when i receive; wait ...seconds; until)	1	3	2	9	8	1	3.96	1.23
5	Variables	2	3	5	3	8	3	3.88	1.54
10	User interface design	0	3	8	7	5	1	3.71	1.08
3	List (arrays)	2	8	2	4	6	2	3.42	1.59

method to teach programming. On the other hand, for intermediate or senior users it was less useful: as one stated, "As I learned programming before, it was just another language to learn those things."

Perception of using Scratch

As indicated from the Table 2, Scratch is generally beneficial for students in this class. 71% agreed that Scratch is helpful to their programming language learning. More than half of them remarked their preference over Scratch. However, people were less positive when associated Scratch with Actionscript: Less people agreed on the #5 statement. Overall, a majority of learners in this class believed that Scratch was helpful to their programming language learning. However, students were hesitant to make affirmative comments regarding its positive impact on learning Actionscript and their prospective usage.

The Social Aspect

Scratch programming language is tightly associated with development of the Scratch Web (Monroy-Hernández, 2009). On the Scratch interface, there are built-in functions which allow the user to share the Scratch project and post it online, which can be perceived as a method for gaining encouragement from the social community. However, in our research study, 18 out of 24 participants admitted statement #3. On the other hand, most of the students did receive assistance from the online community. 15 out of 24 students agreed to some extent with the statement #1. Therefore, the online community still serves as a place where learners can find resources and assistance to complete their Scratch project.

Table 1. Programming concepts in Scratch which are helpful for learners' understanding

#	Statements	SD	D	SD	SA	A	SA	Mean	SD
1	While using Scratch, I got help from the Scratch online community (www.scratch.mit.edu).	1	6	2	4	4	7	4.04	1.71
2	I think overall Scratch is of help to my understanding of programming concepts.	1	8	0	3	5	7	4.00	1.79
3	I didn't share my Scratch project with others in the online community.	1	5	0	3	9	6	4.33	1.61
4	I would love to make some new Scratch projects.	4	4	3	2	8	3	3.63	1.76
5	I would prefer spending more time on Scratch before starting with Actionscript programming.	3	5	3	5	6	2	3.50	1.59
6	I will not use Scratch in the future.	5	4	5	3	3	4	3.29	1.78

Table 2. Students' Perception of using Scratch in this course

Challenges

Although using Scratch has an overall positive impact, seven of the respondents reported that they did not believe that Scratch is conducive to their understanding. Among these respondents, many of them failed to find the association between ActionScript and Scratch due to a variety of reasons. Two of them mentioned that the lack of

syntax in Scratch made it difficult to relate to the ActionScript environment. Another one perceived it difficult to perceive any relevancy between the two: "I had a hard time grasping what the sprite or the program was attempting to do. I had some idea of what ActionScript was supposed to do, but not a clue what Scratch was trying to accomplish." The senior learners certainly were not able to gain more benefits than the novice learners. Besides, another senior programmer commented bluntly that Scratch was useless and time-consuming to him.

Learner style is another notable element concerning the factors that contribute to a positive perception of using Scratch. One of them remarked that "I understood programming concepts by the way that Scratch presents them visually." A non-visual learner may also find it difficult to learn from Scratch, as one noted in his words.

Lastly, some other learners found that a lack of instructions could also hinder the process of learning from Scratch. Scratch may not be as intuitive as the instructor assumed it to be. Another one said Scratch makes it easy for him/her to mix all the functions, methods, and variables together, especially when clear instruction or guidance was lacking.

Discussion and Recommendation

The findings in this study show an overall positive perception of using Scratch for programming learning. Some fundamental concepts are easily understood using Scratch, while complex concepts, are relatively less understandable in Scratch. These findings were paralleled with Maloney et al.'s (2008). With that said, these research findings also suggest some challenges with using Scratch for programming learning. First of all, as many novice learners find it difficult to relate Scratch to command-line programming, it is recommended that instructors should provide explicit guidance and instructions relating to the conceptual structures between the two programming environments for students, especially those at the beginner level. It would have been more helpful to provide materials which direct the learner to walk through some of basic programming concepts.

Second, there is a great need for separating the teaching method among learners in different levels. As indicated in many previous research studies, Scratch is more appropriate for novices, particularly children, who have just started to learn programming (Maloney et al., 2008; Resnick et al., 2009). Therefore, if there are several senior programming learners, it might not be as justifiable to integrate the usage of Scratch. Alternatively, different levels of tasks based on the learner's proficiency with programming language could be provided.

Lastly, more encouragement for sharing one's Scratch project on the online community could enhance the use of the online resource. As Scratch is social in nature (Resnick et al., 2009), students would be able to gain more support if they are more connected to the online community, especially considering the availability of the built-in share function. Furthermore, by being active in the online sphere, learners will be able to not only receive assistance from people online, but also potentially continue to support and collaborate with each other afterwards.

Conclusions

As a graphical environment, Scratch allows programmers to snap together a set of command blocks in different sequences and combinations and structure them in a way in which one programs in a formal programming language. This so-called *tinkerability* not only makes programming more playful and engaging to programmers, but also helps them experiment with new programming concepts incrementally and iteratively.

By and large, the Scratch learning experience did ease the transition to a more advanced scripting language as suggested in this exploratory study. However, challenges remain as it is patent that novice programmers benefited more from this Scratch experience than intermediately skilled programmers. Hence, instructors are recommended to separate the instruction between these two groups, providing explicit and more specific guidance to novice programmers while encouraging intermediate-level programmers to complete advanced-level tasks on Scratch so that potentials of using Scratch can be live up to a full state.

References

diSessa. (2000). *Changing Minds: Computers, Learning, and Literacy.* MIT Press, Cambridge, MA.

Lahtinen, E., Mutka, K.A., & Järvinen, H.M. (2005). A Study of the Difficulties of Novice Programmers. ITiCSE'05, June 27–29, Monte de Caparica, Portugal.

Linn, M.C., & Dalbey, J. (1989). Cognitive consequences of programming insturtion. In E.Soloway & J.C. Spohrer (Eds.), *Studying the novice programmer* (pp. 57-79). Norwood, NJ: Ablex.

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. Retrieved from http://web.media.mit.edu/~mres/papers/sigcse-08.pdf

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. ACM Transactions on Computing Education, November 2010.

Prensky, M. (2001). Digital natives, digital immigrants. *On the Horizon, 9, 5*, 1–6. 16.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer *Science Education, 13(2)*: 137–172.

Resnick, M. , Maloney, J., Hernández, A. M., Rusk, N. , Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for All. Communications of the ACM. Vol 52, No.11.

Rushkoff, D., (2010). *Program or be programmed.* New York: OR Books

Wing, J. (2006). Computational thinking. Communications of ACM. 49, 3, 33–35.

Winslow, L.E. (1996). Programming pedagogy- A psychological overview. SIGCSE Bulletin, 28, 17-22.