Teaching young people computational thinking using MIT App Inventor

by

Raddad Faqihi

B.S., Jazan University, 2014

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2019

Approved by:

Major Professor
Joshua Weese

Copyright

Abstract

MIT App Inventor is a visual programming platform targeted at beginners to develop mobile apps for Android smart devices. It reduces limitations to programming and app development using a block-based language that decreases syntactic mistakes and encapsulates mobile device features into high-level abstractions that are straightforward to incorporate into apps. This study investigates how App Inventor can be integrated into high school computer science and engineering courses to foster computational thinking by developing mobile apps. Teaching fundamental computational thinking concepts and skills to high school students is currently a curricular concentration in various nations. Working in correspondence to this aim are advanced programming environments, curricula, and learning methodologies that intend to make computer science more convenient and motivating. In this study, we explain the design and evaluation of App Inventor, a programming language that aims to help novice programmers develop computational skills through building mobile applications. As observed in this study, App Inventor can be a convenient and powerful platform that could entirely support introductory level courses and foster computational thinking.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am pleased to express my appreciation to Dr. Josh Weese for his guidance, support, encouragement, good humor, and patience throughout the entire study. Also, I would like to thank the members of my thesis committee, Prof. Mitchell Neilsen and Prof. William Hsu for their expertise, excellent insight, and assistance.

My thanks additionally are extended to Saudi Electronic University and Saudi Arabian Cultural Mission for financial support during my graduate career. I would like to express my appreciation and thanks to my family for their support, love, and encouragement, especially my lovely wife, Shaima, who supported me through the challenging periods. Without my wife, completion of this work would not have been possible.

Chapter 1 - Introduction

Computing has become a modern subject in the school curriculum in various countries and a significant part of free learning opportunity in others. Most public schools, especially high schools, need to include the computing field as a requirement not as elective courses in their curriculum because the demand in the field has grown very fast. Moreover, most people have shifted their use of devices to those that are digital and capable of connecting to the internet. However, K-12 schools struggle and find difficulties in including computer science in their curriculum because they lack the number of teachers who have knowledge and experience with computing, or they lack funding (WaldenUniversity, 2019; Yadav A, et al., 2017; Google Inc, Gallup Inc, 2016).

Schools need to prepare the classroom environment to teach computer science so that students are aware of how this field has become essential. Computer science, which is the study of principles and use of computers, has become significantly more important based on rising job market demand and career paths that use technology (UniversityofWisconsin, 2019). Many industries and organizations have been looking for people who have obtained computer science degrees (Johnson, 2015). However, K-12 schools do not usually offer computer science in their curriculum**.** Offering it would allow K-12 students to think about computer science and the possibility to continue studying in this field in post-secondary education (Zimmerman, 2018). The key to engaging computer science in K-12 schools is to include Computational Thinking (CT) concepts in designing and teaching the curriculum of computer science (National Research Council (U.S.), et al., 2011; Denning, Peter J., 2017). It is necessary for instructors to use computational thinking concepts in teaching (Yadav, et al., 2017; Yadav, et al., 2014)because

these concepts will help them in teaching and explaining the most challenging topics in STEM subjects (Yadav., et al., 2017).

Moreover, there are many positions available in the computer science field around the United States. There an estimated 500,000 jobs in computer and information technology. Today, computer science jobs have increased in the last decade and will continue growing for the next decade (Jiang, 2018). Smartphone use has been on the rise as well, particularly in young teens. This gives context for engagement in technology. Schools can take advantage of this trend to motivate students in computer science. In the United States, the rate of mobile computing used by teenagers has been estimated at 88**%** for smartphone adoption, which means there are around 74.2 M teenagers who use smartphones (Lenhart, 2015; Anderson, et al., 2018). Therefore, it would be an excellent opportunity to take advantage of this number to motivate students and concentrate more on computer science courses.

Many people would like to learn or have tried to learn to code but agree that it is not easy. They struggle because of many factors, such as the preconceived notion that it is difficult to understand the syntax of the program languages, or that they are not confident in their abilities in mathematics, logic, and problem-solving skills. Others succeed in learning to code and have found that programming is not as complicated as they thought. Instructors could make computer science more straightforward and comprehensible in colleges and universities if they allowed for earlier enrollment in the courses that are related to the computer science field or technology. Those courses are currently only electives for students in most high schools (Code.org, 2018).

So, what is computational thinking? There are many definitions, but CT can be interpreted as "a method of solving a problem that includes abstraction, algorithmic thinking, and other concepts collected from the study of computer science." (Wing, 2006) Alternatively, it is a

mental process of expressing a problem to allow a solution. The solution can be accomplished by humans, machines, or a combination of both. Computational thinking has four key fundamentals that can help people to understand them. These computational thinking fundamentals are decomposition, pattern recognition, abstraction, and algorithms. If schools integrated or introduced computational thinking concepts into their classrooms, more students would be successful in computer science skills. In this paper, we are going to implement an experiment by using a visual based platform for non-programmers called APP INVENTOR.

App Inventor is a web-based tool created by Google and subsequently maintained by The Massachusetts Institute of Technology (MIT, 2018). Individuals can develop and build apps for mobile devices by using App Inventor platform and release them in Play store. Mobile applications can be designed and developed by arranging components with a "what you see is what you get" WYSIWYG editor in a web browser, where the computer is attached to the device by WIFI or USB (The George WashingtonUniversity, 2019). The role of the components is specified using a blocks-based graphical programming language. An essential point in making App Inventor friendly to novice programmers is live programming; code changes are immediately and continuously displayed in the running program on the devices (Schiller, et al., 2015).

App Inventor is designed to be accessible and engaging to college nonmajors taking introductory classes in computer science (Spertus, et al., 2010). It enables beginners or non-programmers to create applications for the Android operating system. Users can create and develop apps through the visual block-based development environment. By using computational thinking concepts and App Inventor simultaneously, they will learn the principles and fundamentals of programming and coding. MIT is working on the App Inventor’s next phase as

well, which makes it available and reliable for IOS operating system users to create and develops apps for smartphones and tablets that run IOS operating systems (MIT, 2018). We would like to acknowledge what has been stated in the research studies for computational thinking concepts in education curricula throughout the past decade. This thesis shows the literature review, which covers the aims of computational thinking skills in the education curricula. The research concentrates on the meaning and core concepts of computational thinking that can be considered when introducing computational thinking into schools. The focus also includes using App Inventor as a tool to practice/develop and asses computational thinking skills. The motivation for this research project is to find a possible solution that would reduce the stigma and the apprehension students and adults (educators) experience when faced with computer innovation and advanced programming technology. Therefore, it became evident that the following question would advance research in this area: to what extent does App Inventor assist young people with little to no prior programming experience to understand and express computational thinking concepts and encourage participation in computer science?

The work structure of the thesis is organized as follows chapter 2 explains the related work, the comparable articles of distinct authors expressing computational thinking skills in education. Chapter 3, 4, and 5 serves as the focus of this thesis addressing the analysis, result, and discussion of the research.

Chapter 2 - Related Work

App Inventor has been implemented in many schools, summer camps, and workshops in order to improve the curricula and include it in K-12 schools. Morelli et al. believe that App Inventor can be a proper platform for bringing computational thinking into K-12 schools (Morelli, et al., 2010). The experiment they did was to create a team which consisted of two undergraduate students, two high school computer science teachers, and an undergraduate computer science instructor to learn how to use App Inventor. They let the students spend four weeks to teach themselves the fundamentals of App Inventor by using online tutorials that are available on the App Inventor website. The students learned quickly and started building their applications. They kept records of their daily activities on the project's WIKI. The project leader helped them in designing and managing a timeline. When they struggled with problems or bugs, the students solved them mostly by themselves. One of the applications developed by the students had a back-end database code written in Python. With no prior experience in programming, they successfully modified the code to fit their application. Moreover, after the students had finished with their applications, they mentored and helped teachers who joined the project. After two weeks, students and teachers worked together to build a prototype of the "Mall madness app," which is a multi-application framework to teach language, mathematics, and computing skills. The result was successful because the students and the teachers learned, improved, and developed sophisticated apps via App Inventor. They concluded that App Inventor could be implemented in K-12 schools and in college courses to introduce students to learning programming and benefit the development of their problem-solving skills.

App Inventor provides a useful object-oriented framework for creating a visual interface and event-driven control structure. Amber Wagner et al. observed that App Inventor was a solid

basis to assist the students in developing an app and understanding its programming (Wagner A, et al., 2013). Their experiment consisted of two methods: one using App Inventor (block-based) and another using Java (text-based). The total number of students involved was 40, 14 of whom had programming experience from their high school. They covered the basics of App Inventor in two days. The students gave a 30-minute presentation in which they implemented their applications and explained how they solved the problems that they faced in building and developing their applications. Then, they moved to Java to design and develop the first program, which was "HelloPorr app." The students developed an app that required packages from Java Bridge. The authors found that visual blocking language gives a powerful new context for motivating computational thinking compared to the text-based language. Furthermore, Wagner and Amber recommended that teaching a visual language platform followed by Java would allow the students to gain the confidence to develop applications.

Sarah AlHumoud, et al. believed that applying and designing computational thinking concepts using mobile application development would help to attract and familiarize students in Saudi Arabia about the reason why they should consider computer science as a choice for their future path and inform them why they should consider computer science as a durable option (Al Humoud, et al., 2014). These concepts include loops, lists, decision statements, animations, and generating GUIs. App Inventor was introduced to female students during the second week of a summer camp. The authors classified learning material into distinct blocks. However, by the end of the week, it was performed as one project. The students learned the fundamentals of programming, such as variables, input-output, if-else statements, and loops. Since the students had no prior programming knowledge, App Inventor was a prominent tool to develop the final project applications. After they built and developed their apps, the authors opened java code

behind their application and gave a general description of the used codes such as if-else statements and loops. The authors concluded that the project accomplished its goals which were to apply and designed computational thinking concepts using mobile application development, these concepts include loops, lists, decision statements, animations, and generation graphical user interface.

According to Weintrop and Wilensky, the learning of program languages based on hybrid visual blocking and text is helping students in high schools (Weintrop, et al., 2018). They believe that there is a relationship between an interface and novice programmers who work efficiently with visual block language would assist students in learning and improving their programming skills. Pencil.cc, which is a customized version of the pencil code, was created for this experiment. A total of 90 students participated in the study. Those students had no prior experience in programming; neither had they taken any classes related to computing. The first homework assignment was to write a program that picked a random number less than 15 and then printed out every multiple of that number until they reached 100. The students started to click through different categories and dragged and dropped the blocks related to loop and math. Each time students faced errors or struggled in solving the problem, they first read the error message and tried to find a solution. The instructors helped and advised them by troubleshooting and reading the error messages. Students worked hard to solve the most critical bugs which were significant on their own. However, the students asked the instructor to give them a hint for each step. The students who were novices to programming learned many techniques in order to solve the problems as well as to develop mobile applications. Weintrop and Wilensky concluded that the notion of modality, which describes the relationship between an interface and learner and highlight modality, could be used to support beginners in early programming successes.

According to Honig, the App Inventor improves students' skills in computational thinking (Honig, 2013). Honig's experiment, which involved over 100 students, showed that the App Inventor is an effective way of teaching quantitative thinking skills to non-computer science majors. He measured the result by using an assessment process motivated by Bloom's Taxonomy (Johnson, et al., 2007; Christopher, et al., 2008). Honig taught a class that was designed for undergraduate students to meet a university-wide requirement for "quantitative knowledge" learning. He restructured the course to increase its appeal to students while maintaining important learning goals. By using a mobile app which was represented as App Inventor, students would learn quantitative skills. The students learned how to understand the App Inventor by doing assignments that were given by the instructor. They developed their applications at the end of the semester. The result confirmed the effectiveness and the ability to motivate students. Moreover, he concluded that the result was positive in three assessment categories remembering, understanding, and application. Honig continued that App Inventor should become a part of the assessment process for the students. He said, "While the assessment process can be improved to add precision, the results endorse the use of the cognitive categories to assess computer science related learning".

In addition, Shuchi Grover and Roy Pea rely on the Vygotskian theoretical framework, which emphasizes the significance of social interaction in the progress of different mental processes (Grover, et al., 2013). Seven middle school students, three girls, and four boys, who had zero experience in programming, participated in a workshop. The authors collected the data from pre and post surveys, video, and audio recordings. The authors introduced the essential components and features of the App Inventor to the participants so that they could develop their apps. The curriculum was designed to be comprehended through discussion and questions since

the aim of the research was to study discourse intensive pedagogy for introducing students to computational concepts. The students started to discuss their ideas about the apps that they were going to develop. They used new vocabulary and concepts related to computing to help build their apps. When the authors explained and gave an example of the App Inventor application which was “Hello Purr”, the students asked the instructor “what if I, umm just wanted the cat to meow five times?”. This indicated that students were thinking to add more features and tried to implement them. Then, the instructor presented another application that introduced them to increment and decrement numbers and introduced them to understand variables, if-then checks, and conditional statements. The authors believed that App Inventor was excessive motivation for students who are 11 to 14-year-olds to learn to program and enjoy creating and developing apps. They concluded that App Inventor was an excellent tool to encourage children to discuss and express their ideas in developing apps. Moreover, they recommended including App Inventor in the curriculum to introduce programming or computer science subjects in K-12 schools.

Howland and Good believed Flip, which is a visual language platform, could be used by young people to create working scripts and examined improvements in their expression of computational rules (Howland, et al., 2015). They measured participants’ ability specify, the computational concepts correctly before and after using Flip. The authors provided the instructors with the materials that helped them in understanding the basics of Flip. A pretest was administered to all classes that participated in this study before the creation game project began. While the students watched a video of computer games being played, the instructor asked the students to write down the rules in their own words. After the project was completed by the students, the post-test was administered. The authors found that Flip motivated the students by introducing the programming concepts and inspired them to create or develop their games by

using a visual language platform. Flip afforded students with opportunities to expand their ideas and implement them in designing and developing games as well as motivate them to introduce computation and programming.

A similar experiment conducted by Ilenia Fronza, Nabil El Ioini, et al. instructors from MobileDev which was a summer school on mobile development that provided classroom training in software development for mobile devices that helped students to learn the analytical thinking skills they needed in order to design and build their mobile applications by using introducing computational thinking via exercises (Fronza, et al., 2015). The authors spent five days of an experiment for a total of 40 hours on their research. The students who participated were from various schools that concentrate on different areas such as technical, scientific, liberal arts, and linguistics. The students, four females and 15 males, they were excited to develop their apps by using App Inventor. The experiment was divided into three parts, which were theoretical knowledge, learning the practical framework, and application of the practical experience using the practical framework. The instructors then started the experiment by giving a brief summary of the App Inventor. The students were guided to solve simple exercises by following the computational thinking steps, which are data collection, data representation, problem decomposition, abstraction, algorithms, automation, simulation, and parallelization.

Afterward, the students elaborated theoretically on how they designed a solution and concentrated on how they implemented the necessary functionality on a component such as buttons in a graphical user interface. Students were involved in discussions and report about their experience and usage of computational thinking in their schools. After the students obtained the necessary knowledge on how to analyze the problems and designed a suitable solution using app Inventor, they started to work on their final projects which were to develop their apps by using

App Inventor. The results of the MobileDev were excellent. The students enhanced their skills, such as analyzing, implementing, and problem-solving. The students successfully solved the problems that they faced when they built their apps using App Inventor. The authors inferred that the MobileDev summer school showed that students with little or no prior experience in computer science were able to improve their analyzing, implementing, and problem-solving skills. The participants played an active role in analyzing problems, architecting solution, implementing software, and validating them.

Joey et al. maintained that text-enhanced graphical programming environment encouraged students to design and develop their applications (Cheung, et al., 2009). They used the BrickLayer, which is a text-enhanced graphical programming environment. It was designed and implemented for junior high school students by the authors. The BrickLayer was intended to support the programming needs of a wearable computing workshop, which aimed at teaching students basic electric and electronic concepts with programming. Students developed programs to process signals from sensors such as light sensors and accelerometers. They designed and implemented the application by using BrickLayer. While the elementary school students found the Scratch workshop engaging and exciting, the junior high students found it less so. The C programming workshop was determined to be complicated by the senior high students, who needed significant amounts of help to complete their projects. Otherwise, students found the BrickLayer workshop enjoyable, entertaining, and informative. The authors concluded that the students found the graphical programming environment such as App Inventor and Scratch are less challenging and more satisfying. They also found that students were not advanced enough to tackle the scripting languages such as Java and C++.

Shuchi Grover et al. believed that open-ended projects of choice would assist students in developing their applications of introductory programming concepts and computational thinking practices (Grover, et al., 2018). The authors used three methods to collect data. First, they used a rubric to evaluate the projects that were built by students in both Scratch and App Inventor from twenty middle schools. The students developed around 80 App Inventor and Scratch projects which were evaluated by the rubric. The authors collected data from survey forms written by teachers, which contained meta-data associated with each project. Most projects were developed within three to five days by students. The rubric was designed to evaluate students in five general concepts design mechanics, user experience, underlying coding construct, and advanced coding constructs. Second, they counted the number of times a criterion, such as coding constructs, was implemented. Third, they used a three-point scale to evaluate the ability and performance of students' projects. The result showed a much higher percentage of apps were coded in App Inventor compared to Scratch. Since the App Inventor was built for mobile Apps and mobile apps are built more for games and utility functions, the students use App Inventor more than Scratch. The authors found that App Inventor projects had fewer bugs than Scratch projects.

2.1 Research Framework and Approach

The research confirmed my choice of the use of App Inventor. Indeed the analysis of previous research studies showed that students were more capable of using and applying it than other visual based programming languages. We decided to conduct an experiment that would prolong and build upon the research conducted by Sarah AlHumoud, et al. (Al Humoud, et al., 2014; Fronza, et al., 2015; Morelli, et al., 2010). By implementing this experiment for three weeks, we were able to obtain more significant data and explore the advanced features of the App Inventor platform more fully. We chose to conduct the experimentation during a regular

semester of the academic year. It was decided to include the experimental curriculum in the class syllabus.

Most students want to design and develop mobile applications, but they struggle to do so because teachers used text-based language programs to introduce students to programming (Fronza, et al., 2015). We prepared a curriculum to introduce and teach App Inventor in class for three weeks. We explained the features of App Inventor and included them in the curriculum. Most summer camp workshops do not have enough time to explore advanced App Inventor features in their curriculum. App Inventor has many advanced features such as database, GPS, Bluetooth, and maps that are not covered deeply in the existing literature.

By starting with block-based languages, novice programmers will learn the core concept of programming languages such as data analysis, algorithm, and abstraction. Novice programmers who start learning to program with script-based languages such as Java and C++, struggle, confused, and spend more time with the syntactic structure of programming languages. By using App Inventor, students can learn the language quickly without complicated syntax. This gives more time to focus on teaching computational thinking concepts instead of giving drawn out instructions on how to use the language itself. This is important because, like most outreach programs and schools, time to work with students is minimal. By using a script-based programming language to introduce it to novice programmers, it discourages learners from looking into computer science options in school. We found a way to bypass the obstacle of coding and introduce all students to computer science. We chose to use App Inventor to exercise computational thinking because it is an innovative, accessible tool that bypasses the arduous and complicated task of understanding syntax or other programming issues and makes programming an attainable goal.

Teaching computational thinking concepts through programming in K-12 schools is challenging because students become overwhelmed with learning a new syntax language (Weese, et al., 2017). The significance of combining computational thinking into existing curriculum and program is not always a straightforward task, particularly in K-12 schools, where problem-solving skills are traditionally emphasized. Most non-computer science instructors see the importance of teaching computational thinking, though it is frequently challenging to distinguish computational thinking from problem-solving. Therefore, in many cases, they believe that teaching problem-solving is sufficient also to teach computational thinking. Authors Voskoglou and Buckley define problem-solving as "an activity that makes use of cognitive and physical means to overcome an obstacle (problem) and develop a better idea of the world that surrounds us" (Weese, et al., 2017; Voskoglou, et al., 2012). Voskoglou and Buckley also perceive that individuals, even college graduates, struggle to apply theory and problem solutions to practice. The authors continue to describe computational thinking as a hybrid mode of thinking, combining logical, abstract, modeling, and constructive thinking. By integrating critical thinking and existing knowledge, these modes of thinking are crucial in solving real-life technological problems (Voskoglou, et al., 2012).

Chapter 3 - Experiment

This research project and its implementation were based on the educational theory developed by John Dewey. Dewey believed that students learn by doing and that school should represent present life (Gorghiu, et al., 2016; Kolb, 1984). With that in mind, the author conducted the following experiment.

This study was conducted in a public high school in Manhattan, Kansas. The majority of the students at Manhattan High School are white and Hispanic, and the gender distribution is 48% female and 52% male (USNEWS, 2019). Computer science is included in their curriculum as elective credits, and students should take eight credits of elective courses in order to graduate. They teach Java programming, game design, webpage design, and computer applications.

The purpose of selecting this particular school was the students have the fundamental basis of sciences as well as have experience working on the computer. The study targets students who are novice programmers to apply computational thinking approaches. The students who participate in this experiment are enrolled in a programming course that is an elective course in their curricula. Fifteen students with little programming experience were involved in the experiment. Since the workshop was the main part of the research, the students were informed that this entailed participation in a research study. Also, IRB-approved assent and consent were sought from the participants and their parents.

The study was conducted over two classes. We meet students on Monday, Tuesday, Friday from 11:29-12:21 and on Thursday's from 9:20-10:55. The second class meets on Monday, Tuesday, Friday from 10:29-11:21 and on Thursday's from 7:40-9:15. The classes' instructors assisted in preparing the class. Each pupil was provided with a computer that had been organized for the purpose of the study. The pre-survey was administered at the beginning of

the first class as well as the post-survey were conducted at the end of the last class. The participants filled out the pre and post survey. The pre-survey aimed to collect basic information about the student's age, ethnicity, prior experience with programming, and knowledge of mobile phone apps. The post-survey aimed to get what the participants learned during the workshop.

3.1 Curriculum

First, a brief summary of App Inventor and an explanation of the main fundamentals of its interface were given. The App Inventor's interface has two main screens a design screen and block editor. The design screen is basically the area that the programmers design the Interface of the apps. It consists of buttons, labels, text boxes, and images. Also, the design screen has visible and non-visible components that do not appear in a display such as Accelerometer sensor, text to speech, and the location sensor. The non-visible components are not part of the user interface screen. However, they provide access to built-in functions of Android devices. The programmers or the designers can design the interface and use varieties of many features provided in the App Inventor.

In order to design and develop apps in App Inventor, students had to have or create a google account. After the students create a google account and sign in Gmail, they can start to create an app Inventor account by using their Gmail email. The design screen has three main parts. The first part is the component palette which located on the left-hand side in the designer view. It contains buttons which can be clicked on and dragged onto the screen. The palette which has the user interface items such as buttons, checkboxes, and labels and other items, was explained to students. The second part is a design screen which displays a virtual screen of the Android devices. In this part, the developers can design the interface of their apps. The third part is the properties of the screen view. It contains different options, such as background color, font

size, and font type. When the programmers drag and drop the non-visible components in their screen designer, the non-visible components will locate below the screen designer. App Inventor has a tiny database component which allows the developer to use it in their apps. In the block editor screen, the students can drag and drop the blocks to program their apps. Each block is connected to the element on the designing screen and characterized by various properties. There are many built-in blocks, such as the control block.

Days	Activity	Duration	Learning
Day 1	Introductions	5 Min	Icebreakers
	Pre-test	5 min	Create App inventor account
	sign up for Gmail account	5 min	learn about App Inventor.
	Setup an account for App Inventor	3 min	Create AI account
	App inventor's Interface	10 min	Explain AI's Interface
	Project 1: Cat and Dog	20 min	Demonstration
Day 2	Project 2: Calculator		learn Textbox, Label, Buttons, Arithmetic Logic, Sensors, loop, If-else Statement
	Create Textbox	5 min	
	Create a label	5 min	
	Create a button	5 min	
	Layout and screen arrangements	5 min	
	Implement if-else statement	40 min	
Day 3	Project 3: Guessing the number		Introduce variable, message dialog, if-else statement, initialize value, interface design.
	Create a global variable	10 min	
	Use message dialog	20 min	
	Dive more with if-else statement	10 min	
Day 4 to Day 10	Team Projects		Brainstorming, Work as a group, Problem-solving, Discussion
	Work on their Projects		
	Assist students in their projects		
	Answer their questions		
	Submit their Projects		
	Post-test		

Table 1 Syllabus

The first day consisted of an overview of the App Inventor. After the participants finished the pre-survey, the instructor explained and implemented the first app, which was “Hello Purry,” as shown in bellow Figure 1 First App Editor Block. The purpose of the app is to interduce

buttons, images, sounds, and non-visible component. When the user clicks on the cat's button, the mew sounds play. Also, when the user shakes the phone, the sound will pause. Then, the students started to explore the app inventor to familiarize themselves with the design interface and block editor interface. The students were asked to implement the first app on their computers. They started to design the interface; most students begun to draw in a piece of paper and then move to implement it in App Inventor. After they finished the designing, they moved to block editor interface to program the application.

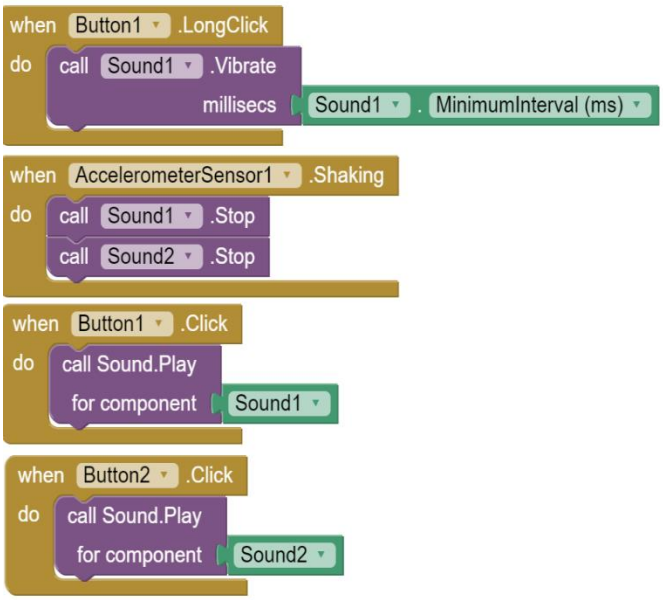

Figure 1 First App Editor Block

On the second day, we explained the variables, user input, and mathematical operations by developing a calculator app, as shown in below Figure 3 Second App Blocking Editor. The purpose of the app that was presented by the author was to introduce mathematical concepts and to learn using labels, textbox, variables, and if-else statement. Since we explained the if-else statement and implemented it in front of students to understand the purpose of using it in developing apps, students used them in their projects. Then, the participants started to design the

interface, then applied the logic in the block editor. After they built the app, the author asked them to implement dividing by zero. The students figured out the solution by using if-else statement and the notifier component. After the students finish implementing their apps, we noticed some students add features to the calculator, such as shaking the phone to clear the screen. This indicated that the students were interested in developing mobile applications.

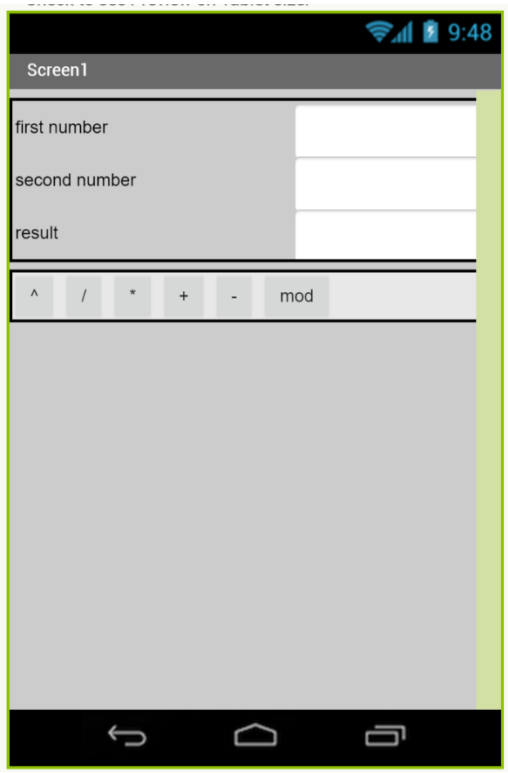

Figure 2 Second App Interface

Figure 3 Second App Blocking Editor

On the third day, the author explained the global variable, which is used to give information among sections of code that do not share caller relation. The third app was guessing the correct number which the most challenging app for the students to implement it, as shown in Figure 4 Guessing Game Block Editor. The purpose of the app was to introduce the concept of a global variable and to practice more on the if-else statement and other programming languages

concepts. The goal behind this game was the computer picks a random number between 1-100 and the player going to guess the correct number. The computer will tell the players if each guess is too high, too low, or the exact number.

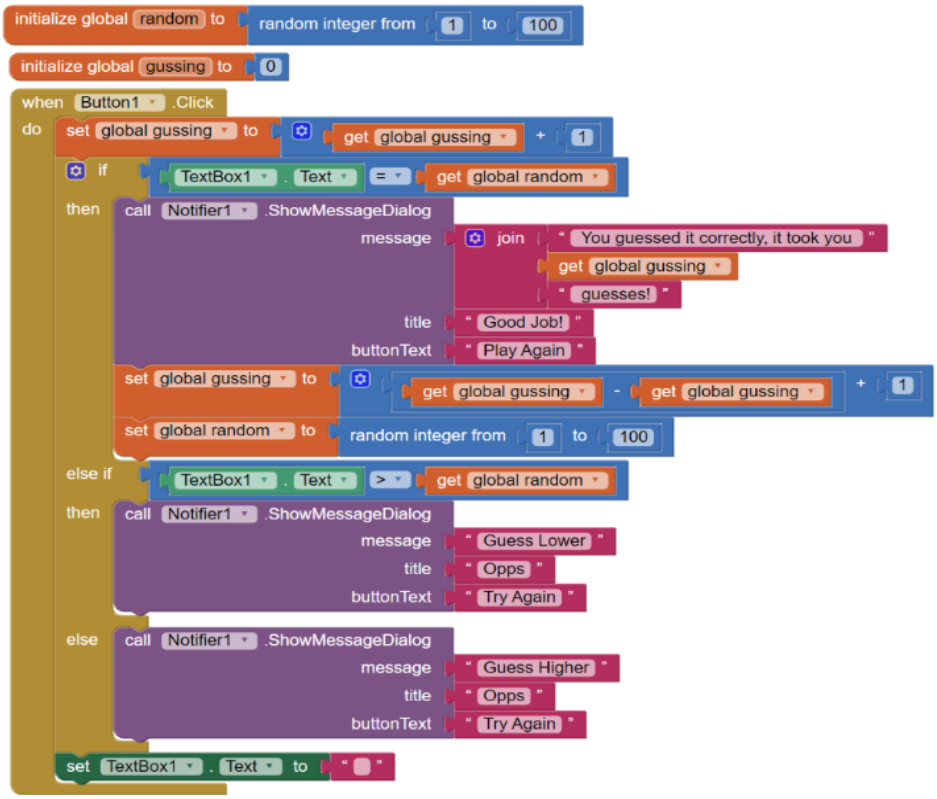

Figure 4 Guessing Game Block Editor

The students were excited to develop this game. The interface consisted of a button, label, and textbox. After the student finished designing the interface, they switched to the block editor. They began to create a variable and named it a random number. Then, they used the if-else statement to compare whether the input number was correct or not. The students used a notifier component to pop up a message showing the input was right or not. After that, they added if-else to know whether the number was too high or too low. Then, they used an else statement to show pop up a message showing that the player guessed the correct number. Since the purpose of this app was to challenge the students and see how they were going to deal with

the problems, the author asked them to calculate how many attempts the player guessed till they guessed the correct number.

The next day, students spent the rest of the time to figure out to solve how to avoid dividing by zero. Most students came up with different ideas to solve the problem. Some participants looked into a similar problem in other applications and tried to do the same solution. This indicates that students have developed their abilities in problem-solving by using or modifying existing solutions.

The fifth day, we assisted students in brainstorming to design and develop their projects. After the students came with ideas about their projects, they started to design the interface. They spent a week in implementing and developing their apps. The author noticed that when students faced problems, they tried to solve them by themselves instead of looking for help from the teachers. They used many techniques in solving the problem, such as breaking down the problem into small pieces and solve them separately. One student faced an issue when implementing the app. The app was a game, and the issue that he faced was the ball moved in one direction, but he wanted the ball to move in all directions. He was troubleshooting the problem by following the block step by step from the beginning, and he found that he did not use the correct variable in the block.

The six and seventh day the classes were canceled due to weather conditions. On the eighth day, Students continued working on their projects, and the author noticed that most students did not need much assisting on their projects because they knew to debug and troubleshooting the problems. Also, they were able to break down the issues into small pieces and solve them. On the next day, the student submitted the post-test survey. Then, they presented their projects in front of a professor from Kansas State University, author, and their teachers.

3.2 Instrument Design

Two instruments were created to evaluate students' computational thinking. The pre-post survey instrument consisted of 23 questions. The first section consisted of five questions about their demographics, such as age, ethnicity, and gender. The second section contained eighteen questions about programming experience, programming languages, and problem-solving self-efficacy.

The questions were adjusted from a project that collaborates between MIT, Wellesley College, et al. Some questions were changed from their original context to fit with this project (MIT, Wellesley College, et al., 2018). Question 7 has changed from "How many program languages have you studied in school" to "Which programming languages have you used?. Question 8 has altered from "Have you developed any applications" to "Have you developed any mobile applications?". The first five questions asked about students' demographics, such as their DOB, gender, and ethnicity. Also, some questions collected information and background in computer programming. The survey had sections and questions categorized by relevant problem-solving confidence, computational thinking concepts, and experience in Science, Technology, Engineering, and Math (STEM). These questions measured self-efficacy on a five-value Likert scale: strongly disagree, disagree, natural, agree strongly agree (Fronza, et al., 2016).

The pre-survey was administered online on the first day before anything was explained about the class. Out of 18 participations, three students were excluded for missing pre-survey. Post-survey was given on the last day. The instructor collected the data from pre and post-survey. Also, in the middle of the workshop, the author collected the students' applications to evaluate them. At the end of the workshop, their projects were collected to assess them. A

Cronbach's Alpha of 0.81 on the pre-survey and 0.70 on the post-survey shows that the survey is reliable.

Chapter 4 - Findings

In order to measure any improvement overall, we observed the mean scores on the pre and post-survey. Both surveys comprised fifteen questions in total. The mean score was 53.4 (SD = 7.2) on the pre-survey and 54.7 (SD = 5.1) on the post-test. A difference was highly significant on a paired samples t-test ($n = 15$, $t = -1.78$, $p < 0.048$). Background information was gathered, such as programming experience as shown below in Figure 5 Programming Experience. Over 33% of students had experienced block-based languages, and more than 32% they are very confident in their ability in science. Over 45% of students had experienced the basics of Python, and over 76% of students had been exposed to basic JavaScript and HTML. This shows that more than 80% of students have used or experienced basic programming languages because they were enrolled in computer science courses and were exposed to computational thinking. Students who had experience in programming languages improved more in computational thinking concepts such as data collection, data representation**,** and algorithms**.**

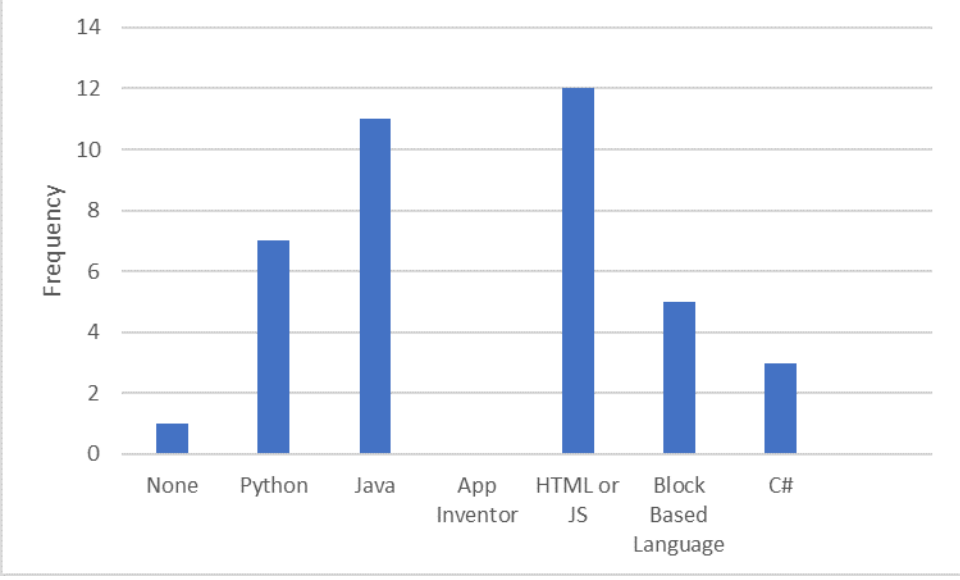

Figure 5 Programming Experience

	% of student	Avg. Pre-Mean	Avg. Post-Mean	Avg. Std-Dev
Programming Skills	33%	2.9	3.2	0.8
Unsolvable situation	27%	3.3	3.6	0.9
Confidence in Math	-33%	4	3.7	0.9

Table 2 Mean of pre-post survey

Students who had more experience in programming languages enjoyed solving programming issues while they developed their mobile application more than students who had little or no experience in programming. Those students who had no expertise showed more confidence in pre-survey than students who had experience in programming. Since they might not have been exposed much to computational thinking, this led to overconfidence, which is causing the amount of improvement observed in the post-survey.

The averages of questions 1-15 pre and post, as shown in Appendix A, indicate improvement in problem-solving skills. Since the experiment implemented on small group consisted of 15 students, the effect size which calculated by using pooled standard deviation As shown in Table 3 Effect Size was not significant because the questionnaire was administered to a small population. However, there were some questions had medium to large effect size, such as question one which focused on programming skills and question ten asked students to rate themselves about their confidence in math. Besides, question thirteen asked students about are they going to take more classes in computer science in the future, and this question had a significant effect size. Since students were excited to develop applications for smartphone, the effect size was positive in the majority of questions except question ten. Since the mean in question ten at pos-survey were smaller than the mean at pre-survey, the effect size was negative. The direction of the effect, whether was positive or negative, depends on the mean in the pre and post-survey.

Students who had experience in programming languages indicated a significantly higher in effect size. Most students stated that they wanted to enroll more in computer science classes in the future.

Questions	Effect Size	Difference Mean	PooledStD
Q1	0.43	33%	0.77
Q2	-0.08	-7%	0.82
Q3	-0.06	-7%	1.08
Q4	0.00	0%	0.66
Q5	0.31	27%	0.87
Q6	0.12	13%	1.15
Q7	0.15	13%	0.91
Q8	-0.11	-7%	0.62
Q9	0.08	7%	0.83
Q10	-0.39	-33%	0.85
Q11	0.22	20%	0.91
Q12	0.30	27%	0.89
Q13	0.40	40%	1.00
Q14	-0.07	-7%	0.98
Q15	0.31	13%	0.43

Table 3 Effect Size

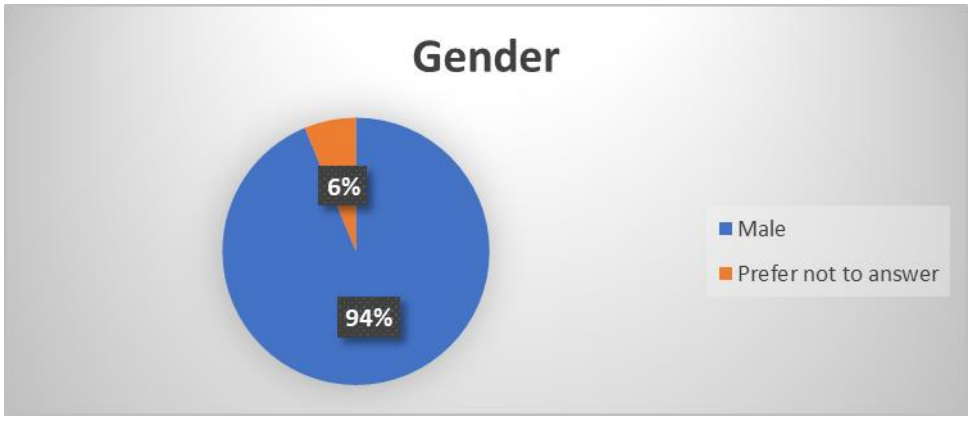

Figure 6 Gender

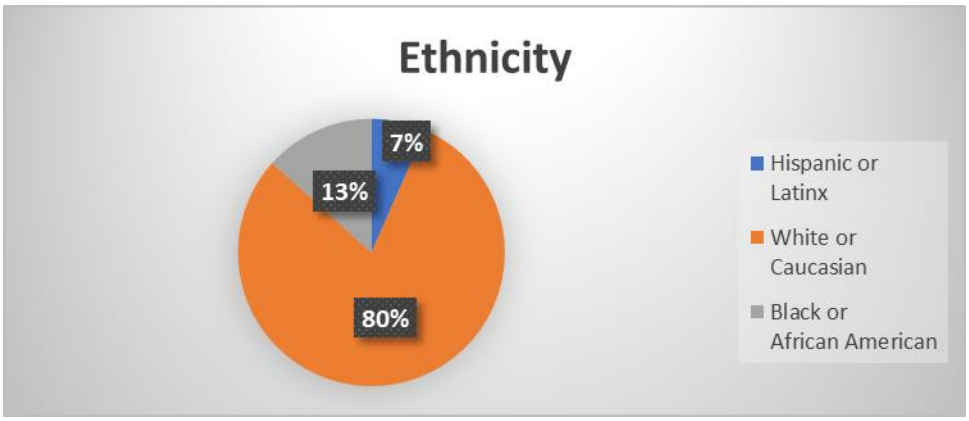

Figure 7 Ethnicity

In addition to looking at absolute differences in mean pre and post-survey scores, we also calculated normalized learning change (Marx, 2007). This takes into account the maximum possible positive or negative learning gain, given the pre-survey score. The normalized gain has been broadly applied in assessing students' performance in pre- and post-surveys. The average can be computed using either the average scores of the class or individual student's scores. In general, these two calculations give different results. The nature of these two results is examined in several idealized conditions. The outcomes recommend that we may be capable of utilizing the variation between two results to derive data on how the population may have altered as a result of instruction (Knight, 2010), which is suitable for circumstances in which there are cases of negative learning gain. Moreover, to look at absolute differences in mean pre-test and post-test scores, the author calculated normalized learning change, which takes into account the maximum possible gain or loss given the pre-test score. Normalized learning change is appropriate for situations in which there are instances of negative learning gain, which was the case for a small number of students. Learning gain is calculated as 100 * (post-pre)/(100 - pre), and a modified calculation is used for students with negative learning gain: 100 * (post - pre/pre). Overall, there

was a positive learning gain per questions of 43% and 30% learning gain per students, which indicated that students improved their skills in computational thinking as shown below Figure 8 Learning Gain Per Questions and Figure 9 Learning Gain Per Students. These results suggest that App Inventor helps in teaching people who have some experience as well as those who have little to no experience in programming (Howland, et al., 2015).

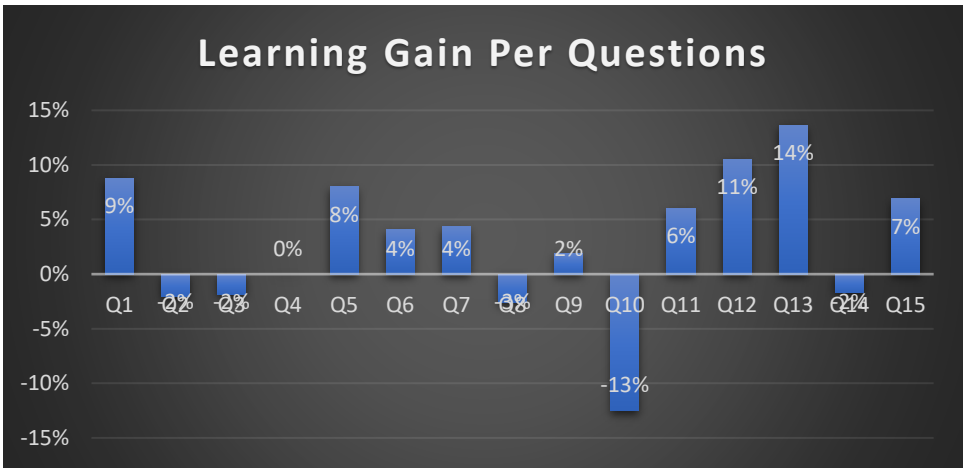

Figure 8 Learning Gain Per Questions

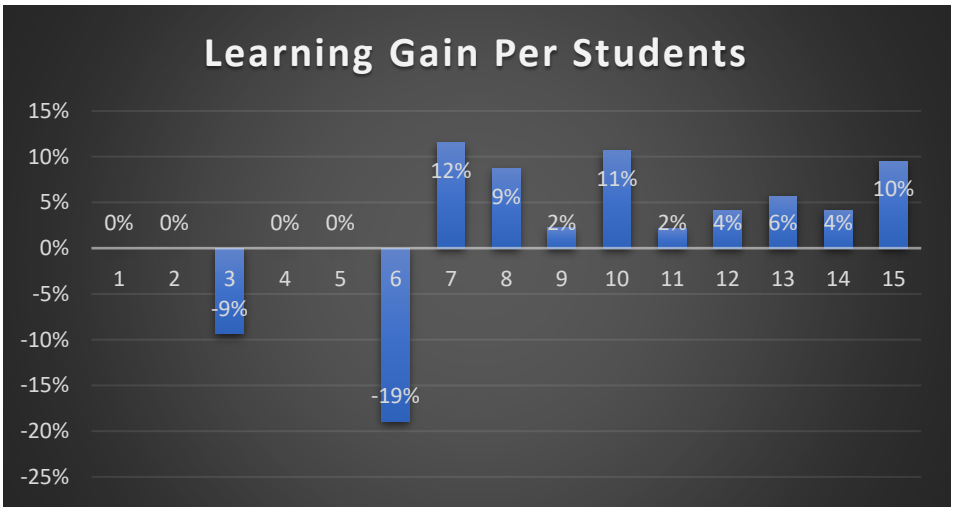

Figure 9 Learning Gain Per Students

4.1 Assessment and Rubrics

The rubric was developed to evaluate the final projects, as shown in table 3. Students were assessed in five categories. First, the ability of students to gather appropriate information in

order to use it in solving a problem, in general, is called data collection. The author evaluates how necessary information was collected by students to implement it in their projects. The second step consisted of data analysis in order to find a pattern and draw conclusions. The third criteria data representation evaluated, is how to organize and display data in relevant graphs, words, or images. The fourth criteria assessed the students' abilities to break down complex tasks into smaller tasks. The last item on the rubric is an algorithm which represents how to generalize a process to solve problems so that it can be applied to other problems. Five students out of fifteen obtained full marks in collecting and Analyzing Data, Problem Decomposition, Algorithm, and Visualize Data. Five students out of fifteen earned 75 out of 100 in their projects because two students did not finish their final projects because their apps were too ambitious and too challenging. The other three students obtained this score because they had no experience in programming. Even though they had no experience in programming, they earned a good score in visualization and collecting data which indicated the improvement in their problem-solving skills. Those students who obtained full mark in their final projects used some advanced features that I did not explain them in the class. This indicated that even the students who had prior experience programming, they improved in self-learning and problem-solving skills because they were able to implement some sophisticated features of app inventor in their applications. When we analyze their final projects, we noticed that they had experience working on programming languages such as python and java, as shown in Table 5 Final Project Score.

RUBRIC COMPUTATIONAL THINKING				
	Poor (10)	Fair (15)	Good (20)	Excellent (25)

Collecting and analyzing data	Cannot collect data systematically	Can collect data systematically but cannot organize and understand it logically	Can collect data systematically and can organize and understand this logically but cannot draw a conclusion	Can collect data systematically and can logically organize and understand it so that a conclusion can be drawn
Problem Decomposition	Can divide a simple task but not a more complex task	Can divide the simple task into logical pieces but not a complicated task	Can divide a complex task into smaller tasks but the components are not logical	Can divide a complex task into logically smaller tasks
Algorithm and Procedure	Cannot generalize a process for solving problems	Can generalize a process to solve problems	Can generalize a process to solve problems but sees no other applications	Can generalize a process to solve problems so that it can also be applied to other problems
Visualize Data	It is not possible to display information in relevant graphs and tables	It is partly possible to display information in relevant graphs and tables	It is possible to display information but the choice for graphs, tables are irrelevant	can display information in relevant charts and tables

Table 4 Rubric

Student	Collecting and Analyzing Data (25)	Problem Decomposition (25)	Algorithm and Procedure (25)	Visualize Data (25)	Total (100)
1	23	22	22	18	85
2	23	22	22	18	85
3	25	25	25	25	100
4	20	17	18	20	75
5	20	17	18	20	75
6	25	25	25	25	100
7	25	25	25	25	100
8	22	23	25	25	95
9	25	25	25	25	100
10	23	20	18	15	75
11	25	25	25	25	100
12	18	18	20	22	78
13	18	18	17	22	75
14	25	25	20	25	95
15	20	17	18	20	75

Table 5 Final Project Score

The first step in designing and developing mobile apps starts by collecting projects' ideas. The participants are free to decide whatever idea they want to develop, and our role is to assist them in understanding the level of complexity and feasibility. While the novice programmers were looking for the projects' ideas, we perceived that most of them were using the Internet to skim existing Android Apps to become inspired. The majority of applications that were developed were a moderate level of complexity that intends to advanced technology and tools such as database, sensors, GPS, Bluetooth, and maps. The functionality of the final projects allows us to acknowledge that the result of the class is positive since the students implement advanced features in their final projects. Despite their experience, students can exercise computational thinking skills to step through the process of distinguishing a problem to solve it in the form of a functional product. App Inventor assists young learners to develop their solutions without requiring specific training on software development tools. Furthermore, students were asked to illustrate the architecture, design, and implementation of their mobile application, revisiting the customized development process that was followed through the workshop sessions. As an outcome, all students demonstrate their products in terms of a set of sequences shown in blocks, which are carried out by the different graphical user interface elements at hand. Notwithstanding being time-consuming, the assessment structure helped in showing conceptual gaps. Moreover, participants were happy to explain their work. Hence, it is worth conducting more experiments to validate the assessment framework. Since students discuss problems from several standing points, brainstorm ideas from various approaches, they improve their skills in analyzing data, and developing mobile applications. Further, the fact that the final product is a mobile application improves the excitement and curiosity of students, as they relate to a range of age that makes heavy use of devices like phones or tablets.

The majority of students felt comfortable learning a new environment of programming, working in teams collaboratively, creating graphical user interfaces, and reading the documentation. After the author analyzed the final projects, he noticed that three students were challenged when they had to break down a complex task into logical pieces. They were not able to decompose a complex task into smaller elements because they had not been exposed to computational thinking. They had no experience in programming languages. However, they obtained a higher score in visualization data because they studied interface design. Students explained their products in terms of a series of sequences shown in blocks, which are carried out by the different graphical user interface elements at hand. Also, more than 65% of students' projects used decision statements, loops, method calls and other advanced components such as Bluetooth, GPS, and Accelerometer Sensor. This indicated how they were exposed to computational thinking by the levels of apps that were presented during the class. Two projects were not completed, which means either the projects were too challenging, or they should have started working on their projects earlier.

Chapter 5 - Conclusion and Future Work

This study was the result of personal experience and thought about the relationship learners have with computer science. We realized that most students or inexperienced persons were discouraged and demotivated with the prospect of using computer science because computer science seemed beyond their reach. We are convinced that the field of computer science needs more people who can teach, simplify, and explain its principles and deliver them effectively and efficiently to students. We believe that there is a path that can make computer science more engaging and attractive in schools. This path involves teaching computer science in schools by implementing computational thinking. Computational thinking is a key to teaching computer science in schools. It gives people the ability and methods to develop solutions to solve problems. Furthermore, we recognized that there is a platform called App Inventor, which can be used to teach novice programmers to build mobile applications. With that in mind, we prepared the curriculum in order to implement an experiment in schools that would test the efficacy of App Inventor.

Both students and teachers were excited to work with a platform that did not require knowledge of programming syntax. After three weeks, students designed, built, and developed their mobile apps. Students were capable of transferring their skills to practice some advanced features of App Inventor such as implementing a database without having to concentrate on syntax and other programming issues.

During this study, we explained the design and implementation of a short-term plan to exercise computational thinking at Manhattan High School. From the first day, students took a role in analyzing problems, developing solutions, implementing software tools, and verifying them. Programming with App Inventor benefited beginner students by letting them concentrate

more on problem-solving and less on language syntax. The satisfaction that the students felt in finding out how to do things and fixing their bugs was visible. Also, observers could recognize that they were achieving confidence and moving toward more significant problems.

App Inventor's built-in emulator indicates that only a few actual phones would be required to introduce App Inventor at the K-12. However, as smartphones become less expensive, they will become more available, and it will, therefore, be easier to support such instruction. Initiatives like MobileDev strive to build a proper environment to practice and implement computational thinking skills, encouraging participants to develop a compelling product that they may think very proudly as their making. Jeffrey Schiller et al. believed that App Inventor assists people with little or no experience in programming to create mobile apps. Ralph Morelli, et al. states that App Inventor would be a proper platform for conducting computational thinking to K-12 students (Schiller, et al., 2015; Morelli, et al., 2010). Our studies agree that the App Inventor would be a suitable platform for bringing computational thinking to K-12 schools.

The outcome of this experiment is supported by previous research such as (Schiller, et al., 2015; Morelli, et al., 2010). As a result of this experiment, we were pleased to note that the participating instructors were delighted to learn about this method, and they decided to continue working on it instead of script-based programming languages. In the end, the results encouraged and motivated the participants to continue working with students and instructors in schools.

Bibliography

Al Humoud, et al. (2014). Using App Inventor and LEGO mindstorm NXT in a summer camp to attract high school girls to computing fields. *IEEE Global Engineering Education Conference, EDUCON*.

Anderson, et al. (2018). Teens, social media & technology 2018. *Pew Research Center*, 31.

Cheung, et al. (2009). Filling the Gap in Programming Instruction: A Text-enhanced Graphical Programming Environment for Junior High Students.

Cheung, J., Ngai, G., Chan, S., & Lau, W. (2009). *Filling the Gap in Programming Instruction: A Text-enhanced Graphical Programming Environment for Junior High Students.*

Christopher, et al. (2008). Bloom's Taxonomy Revisited: Specifying Assessable. *ACM Technical Symposium on Computer Science Education*, 576.

Code.org. (2018). *State of Computer Science Education.* Retrieved from Code.org: https://code.org/files/2018_state_of_cs.pdf

Denning, Peter J. (2017, 2 1). Computational Thinking in Science. *American Scientist*, 13. Retrieved from American Scientist.

Fronza I,et al. (2017). Teaching Computational Thinking Using Agile Software Engineering Methods. *ACM Transactions on Computing Education*, 1-28.

Fronza, et al. (2015). Students Want to Create Apps. *Proceedings of the 16th Annual Conference on Information Technology Education - SIGITE '15*. doi:10.1145/2808006.2808033

Fronza, et al. (2016). Computational thinking through mobile programming a case study in a liberal education context. *Springer Verlag*, 67-80.

Google Inc, Gallup Inc. (2016). Trends in the State of Computer Science in U.S. K-12 Schools. *http://goo.gl/j291E0*.

Gorghiu, et al. (2016). Applications of Experiential Learning in Science Education Non-Formal Contexts. 320-326.

Grover, et al. (2013). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*.

Grover, et al. (2018). What We Can Learn About Student Learning From Open-Ended Programming Projects in Middle School Computer Science. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. doi:10.1145/3159450.3159522

Honig. (2013). Teaching and assessing programming fundamentals for non majors with visual programming. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13*.

Howland, et al. (2015). Learning to communicate computationally with Flip: A bi-modal programming language for game creation. *Computers and Education*.

Jiang, e. a. (2018, 5 2). *pewresearch*. Retrieved from pew research Center: https://www.pewresearch.org/fact-tank/2018/05/02/millennials-stand-out-for-their-technology-use-but-older-generations-also-embrace-digital-life/

Johnson, et al. (2007). Is Bloom's taxonomy appropriate for computer science? *Association for Computing Machinery (ACM)*, 120.

Johnson, P. (2015, 7 9). *ITworld*. Retrieved from ITworld: https://www.itworld.com/article/2945674/computer-science-students-are-in-demand-and-they-know-it.html

Knight, J. (2010). Biology concept assessment tools: design and use. *Microbiology Australia*.

Kolb, D. (1984). Experiential Learning: Experience As The Source Of Learning And Development.

Lenhart, e. a. (2015, 4 9). *A Majority of American Teens Report Access to a Computer, Game Console, Smartphone and a Tablet*. Retrieved from Pew Research Center: https://www.pewinternet.org/2015/04/09/a-majority-of-american-teens-report-access-to-a-computer-game-console-smartphone-and-a-tablet/

Marx, J. a. (2007). Normalized change. *American Journal of Physics*, 75-87.

MIT. (2018, 5 10). *App Inventor*. Retrieved from MIT App Inventor: http://appinventor.mit.edu/explore/index-2.html

MIT, Wellesley College, et al. (2018, 11 20). *Evaluation Tools*. Retrieved from Computational Thinking through Mobile Computing: https://nsfmobilect.wordpress.com/evaluation/

Morelli, et al. (2010). Can Android App Inventor Bring Computational Thinking to K-12? Trishan de Lanerolle.

National Research Council (U.S.), et al. (2011). Report of a workshop of pedagogical aspects of computational thinking. *National Academies Press*, 162.

Schiller, et al. (2015). Live Programming of Mobile Apps in App Inventor. *Publisher: Association for Computing Machinery (ACM)*, 1-8.

Spertus, et al. (2010). Novel approaches to CS 0 with app inventor for android. *ACM technical symposium on Computer science education*, 325-326.

The George WashingtonUniversity. (2019, 5 9). *What You See Is What You Get (WYSIWYG) Editor*. Retrieved from Online Strategy: https://onlinestrategy.gwu.edu/what-you-see-what-you-get-wysiwyg-editor

UniversityofWisconsin. (2019, 5 10). *Computer Science Jobs and Career Outlook*. Retrieved from University of Wisconsin Applied Computing: https://appliedcomputing.wisconsin.edu/about-applied-computing/computer-science-jobs/

USNEWS. (2019, 2 15). *Manhattan High School West/East Campus*. Retrieved from U.S.NEWS: https://www.usnews.com/education/best-high-schools/kansas/districts/manhattan-ogden/manhattan-high-school-west-east-campus-8077

Voskoglou, et al. (2012). Problem Solving and Computational Thinking in a Learning Environment. *Egyptian Computer Science Journal*, 28-46.

Wagner A, et al. (2013). Using app inventor in a K-12 summer camp. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*.

WaldenUniversity. (2019, 5 10). *Top Challenges Facing K-12 Computer Science Education*. Retrieved from Walden University : https://www.waldenu.edu/programs/education/resource/top-challenges-facing-k-12-computer-science-education

Weese, et al. (2017). Assessing Computational Thinking and Problem. *American Society for Engineering Education*.

Weintrop, et al. (2018). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM 49.3*, 33-35.

Yadav A, et al. (2017). Expanding computer science education in schools: understanding teacher experiences and challenges. *Computer Science Education*, 235-254.

Yadav, et al. (2014). Computational Thinking in Elementary and Secondary Teacher Education. *ACM Transactions on Computing Education*, 1-16.

Yadav, et al. (2017). Computational thinking for teacher education. *Communications of the ACM*, 55-62.

Yadav., et al. (2017). *Computational Thinking in Teacher Education.* Rich P., Hodges C: Springer, Cham. doi:https://doi.org/10.1007/978-3-319-52691-1_13

Zimmerman, E. (2018, 9 19). *EdTech*. Retrieved from EdTech: https://edtechmagazine.com/k12/article/2018/09/k-12-schools-work-incorporate-computer-science-curriculums

Appendix A - Pre-Survey

Pre-workshop Survey

This study aims to assist young K-12 students' (9th-12th grade) in learning programming by using MIT App Inventor and develop their computational thinking skills. Also, it will allow us to evaluate the effectiveness of the MIT App Inventor curriculum on student self-efficacy, computational thinking, and general knowledge in computer science and STEM

1. What is your full name? *

Short answer text

2. Date of Birth? *

Month, day, year

3. I am: *

- Male
- Female
- Transgender
- Prefer not to answer
- Other...

4. I am..(choose all that apply) *

- Hispanic or Latinx
- White or Caucasian
- Black or African American
- Native Hawaiian or Other Pacific Islander
- Asian or Asian American
- American Indian or Alaska Native
- Middle Eastern or North African
- Prefer not to say
- Other...

5. Have you taken courses related to computer science (other than this course)? *

- Yes
- No

6. How would you rate your programming skills? *

	1	2	3	4	5	
Poor	○	○	○	○	○	Excellent

7. Which programming languages have you used? *

- [ ] None
- [ ] Python
- [ ] Java
- [ ] App Inventor
- [ ] HTML or JS
- [ ] VB
- [ ] Block Based Languages(Scratch, Lego, NXT, Alice)
- [ ] C#
- [ ] Other...

8. Have you developed any mobile applications? *

- ( ) Yes
- ( ) No

9. Rate your interest in developing mobile applications. *

- ( ) Not at all
- ( ) A little
- ( ) Somewhat
- ( ) Substantially
- ( ) Greatly

10. How interested are you in releasing your own mobile application on the Android Play Store? *

- ( ) Not at all
- ( ) A little
- ( ) Somewhat
- ( ) Substantially
- ( ) Greatly

11. I am confident in my problem-solving abilities *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

12. I enjoy being challenged by seemingly unsolvable situations or problems. *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

13. I am not afraid to ask for help *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

14. Rate your confidence in your abilities in Science *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

15. Rate your confidence in your abilities in Technology *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

16. Rate your confidence in your abilities in Engineering *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

17. Rate your confidence in your abilities in Math *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

18. Compared to your classmates, rate your knowledge of Computer Science *

- Poor
- Below Average
- Average
- Above Average
- Excellent

19. I will take more Computer Science courses after this one. *

- Definitely Not
- Probably Not
- Possibly
- Probably
- Definitely

20. For class projects, I prefer to *

- Work alone
- Work with a group

21. After I graduate from high school, I would like to persue a Computer Science degree. *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

22. I would like to major in Computer Science, but I do not have the skills to do so. *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

23. Computer Science is beneficial to society. *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

Appendix B - Post-Survey

Post-workshop Survey

This study aims to assist young K-12 students' (9th-12th grade) in learning programming by using MIT App Inventor and develop their computational thinking skills. Also, it will allow us to evaluate the effectiveness of the MIT App Inventor curriculum on student self-efficacy, computational thinking, and general knowledge in computer science and STEM

1. What is your full name? *

Short answer text

2. How would you rate your programming skills? *

	1	2	3	4	5	
Poor	○	○	○	○	○	Excellent

3. Which programming languages have you used? *

- [ ] None
- [ ] Python
- [ ] Java
- [ ] App Inventor
- [ ] HTML or JS
- [ ] VB
- [ ] Block Based Languages(Scratch, Lego, NXT, Alice)
- [ ] C#
- [ ] Other...

4. Rate your interest in developing mobile applications? *

- ○ Not at all
- ○ A little
- ○ Somewhat
- ○ Substantially
- ○ Greatly

5. How interested are you in releasing your own mobile application on the Android Play Store? *

- ○ Not at all
- ○ A little
- ○ Somewhat
- ○ Substantially
- ○ Greatly

6. I am confident in my problem-solving abilities *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

7. I enjoy being challenged by seemingly unsolvable situations or problems. *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

8. I am not afraid to ask for help *

- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

9. Rate your confidence in your abilities in Science *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

10. Rate your confidence in your abilities in Technology *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

11. Rate your confidence in your abilities in Engineering *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

12. Rate your confidence in your abilities in Math *

	1	2	3	4	5	
Not confident at all	○	○	○	○	○	Very Confident

13. Compared to your classmates, rate your knowledge of Computer Science *

- ○ Poor
- ○ Below Average
- ○ Average
- ○ Above Average
- ○ Excellent

14. I will take more Computer Science courses after this one. *

- ○ Definitely Not
- ○ Probably Not
- ○ Possibly
- ○ Probably
- ○ Definitely

15. For class projects, I prefer to *

- ○ Work alone
- ○ Work with a group

16. After I graduate from high school, I would like to pursue a Computer Science degree. *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

17. I would like to major in Computer Science, but I do not have the skills to do so. *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly Agree

18. Computer Science is beneficial to society. *

- ○ Strongly disagree
- ○ Disagree
- ○ Neutral
- ○ Agree
- ○ Strongly agree