DEVELOPING INSTRUCTIONAL STRATEGIES AND RECOMMENDATIONS FROM AN INTRODUCTORY PROGRAMMING COURSE IN HIGHER EDUCATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KADİR YÜCEL KAYA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER EDUCATION AND INSTRUCTIONAL TECHNOLOGY

SEPTEMBER 2018

Approval of the thesis:

DEVELOPING INSTRUCTIONAL STRATEGIES AND RECOMMENDATIONS FROM AN INTRODUCTORY PROGRAMMING COURSE IN HIGHER EDUCATION

submitted by **KADİR YÜCEL KAYA** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Education and Instructional Technology Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ____________

Assoc. Prof. Dr. Ömer Delialioğlu
Head of Department, **Computer Edu. & Inst. Tech.** ____________

Prof. Dr. Kürşat Çağıltay
Supervisor, **Computer Edu. & Inst. Tech., METU** ____________

Examining Committee Members:

Assist. Prof. Dr. Halil Ersoy
Computer Edu. & Inst. Tech., Baskent University ____________

Prof. Dr. Kürşat Çağıltay
Computer Edu. & Inst. Tech., METU ____________

Prof. Dr. Soner Yıldırım
Computer Edu. & Inst. Tech., METU ____________

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering, METU ____________

Assist. Prof. Dr. Erman Uzun
Computer Edu. & Inst. Tech., Mersin University ____________

Date: 19.09.2018

ABSTRACT

DEVELOPING INSTRUCTIONAL STRATEGIES AND RECOMMENDATIONS FROM AN INTRODUCTORY PROGRAMMING COURSE IN HIGHER EDUCATION

Kaya, Kadir Yücel
Ph.D., Computer Education and Instructional Technology
Supervisor: Prof. Dr. Kürşat Çağıltay

September 2018, 268 pages

Purpose of this study is to design and develop an introductory programming course for higher education level and extract instructional strategies and recommendations. The course was offered as a visual programming course at the Department of Computer Education and Instructional Technology in Middle East Technical University. MIT App Inventor (a visual programming environment to develop applications for Android OS) was used in the course. The course was 14 weeks long. Basic concepts of programming were offered through a project and product focused introductory programming course.

Design-Based Research methodology was used as the research framework of the study. Under this framework, qualitative data were collected through interviews, observations, and documents. Data were collected iteratively to reshape the design of the course and the instructional strategies until it is appropriate and substantial. Data collection were 2 semester long which included observations throughout the course, interviews at the end of each semester, examination of discussions and products of the students.

Results of this study aimed to provide an instructional prescription for the instructors who are to develop an introductory programming course. An effective, efficient, and motivating course design could help both learners and instructors for the first step of programming education which could lead to an advanced level programming education and help students to grasp computational thinking.

Keywords: Programming Education, Visual Programming, App Inventor

ÖZ

YÜKSEKÖĞRETİM İÇİN GİRİŞ SEVİYESİNDE BİR PROGRAMLAMA DERSİNDEN ÖĞRETİM STRATEJİLERİ VE ÖNERİLERİ GELİŞTİRİLMESİ

Kaya, Kadir Yücel
Doktora, Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü
Tez Danışmanı: Prof. Dr. Kürşat Çağıltay

Eylül 2018, 268 sayfa

Bu çalışmanın amacı yükseköğretim seviyesindeki deneyimsiz programlama öğrencileri için bir öğretim stratejileri ve önerileri tasarlayıp geliştirmektir. Stratejiler Orta Doğu Teknik Üniversitesi, Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümündeki bir görsel programlama dersinden çıkarılacaktır. Ortam olarak App Inventor (Android işletim sistemine uygulamalar geliştirmek için bir görsel programlama ortamı) kullanılmıştır. Araştırmacı teoriyi geliştirmek için 14 haftalık bir ders tasarlamıştır. Temel programlama kavramları, proje ve ürün odaklı bir giriş seviyesi programlama dersi aracılığıyla sunulmuştur.

Çalışmada çerçeve olarak tasarım tabanlı araştırma yöntemi kullanılmıştır. Bu çerçeve altında, nitel veri görüşmeler, gözlemler ve dokümanlar aracılığıyla toplanmıştır. Veri yinelemeli olarak toplanarak, ders ve teori tasarımı uygun ve sağlam hale gelene kadar tekrar şekillendirilmesi sağlanmıştır. Veri toplama süreci 2 dönem boyunca sürmüştür. Veriler ders boyunca yapılan gözlemler, her dönem sonunda yapılan görüşmeler, ve tartışma yazışmalarını ve öğrenci ürünlerini inceleyerek toplanmıştır.

Bu çalışmanın sonucunda giriş niteliğinde programlama dersleri için reçete sunan bir öğretim tasarımı geliştirmek amaçlanmaktadır. Etkili, verimli ve motive edici bir öğretim teorisinin öğrencilere hem de öğretmenlere programlama eğitiminin ilk adımı için yardımcı olabileceği; bunun da ileri düzey programlama eğitimine yönlendirmeyi ve bilişimsel düşünceyi anlamasını sağlayacağı düşünülmektedir.

Anahtar Kelimeler: Programlama Eğtimi, Görsel Programlama, App Inventor

To My Family…

ACKNOWLEDGMENTS

First of all, I would like to thank Prof. Dr. Kürşat Çağıltay for supporting me, teaching me, believing in me, and showing me the way as my advisor throughout my Ph. D. process. I am also lucky to have you as a mentor, a role model, and a friend.

I would like to express my gratitude to my thesis monitoring committee members Prof. Dr. Soner Yıldırım, and Assoc. Prof. Dr. Sinan Kalkan for their invaluable support and feedbacks. I was also lucky to have Dr. Erman Uzun, and Dr. Halil Ersoy in my thesis examination committee.

I would like to thank all of the faculty members in METU and Dokuz Eylul University. Especially, Prof. Dr. Ercan Akpınar, Elif Özyenginer, and Dr. Ugur Altunay. I am grateful to all of the academic staff of CEIT in Kastamonu University, especially, Assoc. Prof. Dr. Göksal Bilgici, Dr. İsmail Yıldız and Dr. Ekmel Çetin, Funda Çatan, Zeynep Piri, and Semih Piri. I am also lucky to have such wonderful friends and colleagues. Sezin Eşfer and Mustafa Güleç were both shareholders of this study. Gülenay Vardarlı, Ersin Kara, Gökçen Aydın, Filiz Çiçek, Mehtap Özen, Mithat Çiçek, Esra Kavasoğlu, Eymen Canatan, and my brother from another mother Durmuş Kuş, knowing to have you all strengthen me with your unconditional support.

I would like to thank Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me and this study financially throughout my Ph. D. education, both in Turkey (2211) and USA (2214-a). I am sincerely grateful to TÜBİTAK. I would like to thank Council of Higher Education (YÖK) for supporting this study through ÖYP project.

I would like to thank my life long supporters, my mother Hatice Kaya, my father Ertuğrul Kaya, and my sister, Pınar Kaya. I am also grateful to have Fevzi Keskin, Vedat and Hülya Tısoğlu, Sinem and Berat Şimşek.

I would like to thank my wife Dr. Seçil Tısoğlu Kaya. We have been through a lot through our Ph. D. education. It would not be finished without your constant love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI:	App Inventor
BA:	Business Administration
BUA:	Bottom-Up Approach
CEIT:	Computer Education and Instructional Technology
CS:	Computer Science
DBR:	Design-Based Research
EME:	Elementary Mathematics Education
ESE:	Elementary Science Education
TDA:	Top-Down Approach

CHAPTER 1

INTRODUCTION

This chapter provides an introduction to the topic, problem, and the study. It consists of background, purpose, significance of the study, research questions, assumptions, limitations, delimitations, and definition of terms.

1.1 Background of the Study

Computing technology is influencing every people's life, thinking, and behaviors, and it is growing more popular day by day (Kazimoglu, Kiernan, Bacon, & MacKinnon, 2011; Kwon, Yoon, & Lee, 2011; Wong, 2017). From the last years of the 20th century to recent years, computer and mobile software technologies get more popular among people. Despite its growing popularity, technology is only a product for the most of the people. Even the people claiming to be experienced about technology (and so-called digital natives) are just consumers rather than producers (H. Gardner & Davis, 2013; Smutný, 2011). As the demand is growing bigger for producers, society needs more programmers to meet the need. However, programming is a very complex task to teach and learn; and it requires special attention (Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011; Kwon et al., 2011). When it comes to inexperienced learners, programming language education is one of the most serious issues in software development (Saito & Yamaura, 2013). Teachers are also interested in how novices learn to program (Rountree, Robins, & Rountree, 2013).

There is a demand for programmers in the industry more than ever. Among young people, interest in programming has grown rapidly. However, when it comes to learning programming and developing software, programming is seen to be a difficult, and complicated topic by the same people. Even the university students (including computer science majors) could get confused by the complexity of programming which could lead to high failure rates in introductory programming courses (de Raadt, 2007; Govender, 2009). Due to this perception and other factors, computer programming courses have a high drop-out rate, up to 50% (Ma, Ferguson, Roper, & Wood, 2011; Porter, Guzdial, McDowell, & Simon, 2013; Robins, Rountree, & Rountree, 2003). According to Jones and Burnett (2008), even the students who perform well in other subjects may not be successful in programming. Obviously, there is a problem with programming education. Since 70s variety of solutions offered to solve this problem (Sorva, Karavirta, & Malmi, 2013). Educational programming environments such as microworlds, visualization software, and visual programming languages are among those solutions (Malliarakis, Satratzemi, & Xinogalos, 2013; Rolandsson, 2013). One of the potential solutions is visual programming languages. Visual programming language was described by Smutný (2011) as “a programming language that lets users create programs by manipulating program elements graphically rather than specifying them textually” (p. 358). The earliest and the best known visual programming language is Logo (Papert, 1993). Since Logo, many visual programming languages are developed to help people learn programming. In recent years, Alice, Scratch, BlueJ, and App Inventor (AI) are more popular than others. Especially, AI provides an easy way to create their own mobile software programs that could help adult novice programmers while others are mainly for K-12 and education.

In visual programming, learners can create their programs just by experimenting with the environment. Even though visual programming environments provide an easier way to learn programming, there is not one true answer for everyone and every situation (Sorva, Lönnberg, & Malmi, 2013). Especially when complex programs are aimed to be created, experimenting and drag-and-drop may not be enough for users. There is a need for instructional guidelines for both learners and teachers for learning programming via visual programming. While there are many studies of expert programmers, studies about novice or non-programmers do not have the equal amount of studies (Robins et al., 2003). Additionally, computer scientists and cognitive scientists have a lot of research studies about programming education, instructional technology has fewer studies about that subject (Kazimoglu et al., 2011; Sajaniemi, 2008). Reiser (2002) defines Instructional Technology as “the analysis of learning and performance problems, and the design, development, implementation, evaluation, and management of instructional and non-instructional processes and resources intended to improve learning and performance in a variety of settings, particularly educational institutions and the workplace" (p. 12). Instructional Technology could provide valuable contributions to the Computer Science (CS) field, especially for the novice learner education. Suggestions should be made by teachers and students to improve the programming courses (Sampaio & Sampaio, 2012).

The programming environment is one of the key components of the introductory programming courses. Learning programming is a demanding task that requires computational thinking to describe and solve a problem then design and code in order to convert solution into the syntax of a programming language (Kazimoglu et al., 2011). Traditionally, students learn programming language's syntax and commands before designing algorithms which could cause a great cognitive load to novice learners. Computer scientists developed many techniques to alter this difficulty. Over the years many programming languages have been developed for novice users to reduce the cognitive load (Wilson, Hainey, & Connolly, 2012). Educational programming environments could be used to help learners to improve their computational and algorithmic thinking abilities. Wing (2006) defines computational thinking as an analytic approach to problem-solving, understanding human behavior by drawing on the concepts fundamental to computer science. While the main aim for the introductory programming courses is to cause students to gain computational thinking and problem-solving skills, much of the introductory textbooks and courses focus on the knowledge about a particular programming language (Robins et al., 2003). Kafai and Burke (2013) stated that "students should not only know about programming but also to think more systematically to solve all types of problems" (p. 65). Moreover, Winslow (1996) states that novice learners approach programming line by line rather than meaningful chunks or structures. This perspective could be fixed by extracting guidelines to develop and design a course with the use of a visual programming environment in practice.

1.2 Purpose of the Study

There are lots of empirical studies about learning programming, however, they are mostly focused on learning outcome during or after the course (Bennedsen & Caspersen, 2012). This study is sought to create instructional guidelines and strategies by reshaping an introductory programming course iteratively, based on the data collected from students by constant observations, interviews, documents and with the support of literature. Purpose of this study is to create instructional strategies and recommendations for an introductory computer programming course via a visual programming language through Design-Based Research framework. Qualitative data were collected through interviews, observations, and documents to reach this aim. Each week of the course, positive and negative implementations were examined to reshape the course. Before selecting the environment, different programming environments were taken into consideration. The target group of this course was the first time learners or novice programmers. According to the literature review, among the textual and visual programming environment, visual programming environments hold the potential for motivating students and have a steeper learning curve. Popular visual programming environments were examined to be chosen as they could be more suitable environments for the first time learners. There were Scratch, Alice, Kodu, and App Inventor as the popular and relatively modern environments. App Inventor was chosen among them considering the target group who were higher education students. Products of App Inventor are working applications for Android OS. Users can develop games and applications with App Inventor just by dragging and dropping "Components" and "Blocks". Interviews were analyzed to see the influence of the visual programming environment, specifically App Inventor. During the interviews, the researcher asked questions about both about visual programming environments and App Inventor environment, specifically. Perception of students towards textual programming environments was also asked the students with no previous experience. Moreover, comparison of the two environments was asked to the students who have taken a textual programming course before.

1.3 Significance of the Study

The main significance of this study is presenting help for the programming instructors by providing instructors some concrete and grounded instructional strategies and recommendations. Programming literacy, which may increase critical-thinking and problem-solving abilities, is "essential today's heavily computer-dependent society" (Wright, Rich, & Leatham, 2012, p. 8). In campaigns like "Code Hour", people from every age group and every sector are encouraged to learn programming. The campaign aims to direct people toward programming not only for make them programmers but also to provide them a new perspective which is called computational thinking. As it was mentioned before, computational thinking is a crucial skill for the 21st century. One of the significance of this study is that results of this study could guide instructors to teach and learners to learn problem-solving, computational thinking and overcome their fears towards computer programming. This would not only change the students' thinking about programming but also will change their perspective towards any problem they will encounter during their daily lives. The common saying "reading the word is reading the world" is now changing to "reading the code is reading today's and tomorrow's world" (Kafai & Burke, 2013).

There is also a more practical (*de facto*) significance of this study. Since the environment is easy-to-use and understand, its products are useful, and with the appropriate instructional theory, it could motivate student intrinsically to learn programming and to be a programmer. While this study is limited to university students, future studies could expand it to K-12 level or adult people. In this way, say, teachers can develop their own programs for their courses which will be a need in near future. Better yet, App Inventor (AI) which is created to develop mobile applications, is getting more popular in last years for both daily use and education. "The mobile learning applications can offer assistance and enhancement of traditional learning, synchronous interactions between the students and teachers, and many ways of interacting with course content." (Bertea, 2011, p. 2). Additionally, this could be a step for advanced programming learning and traditional programming language learning.

Another significance of this study is its methodology. This study uses a Design-Based Research (DBR) framework. Apiola and Tedre (2012) states that there is a need for more action research and formative evaluation which are similar research frameworks to DBR for programming courses. Starting to shape the course through the First Principles of Instruction, and enhancing it through the experience are other significant parts of this study.

Finally, this study could enlighten the programming education problems and solutions for Turkish context. Teaching and learning programming is found challenging worldwide, however, it is crucial to understand local practices of the topic (Apiola & Tedre, 2012).

1.4 Research Questions

Research question of this study and sub-questions are:

> *What are the instructional strategies and recommendations to develop an efficient, effective and engaging introductory programming course for non-CS majors?*
>
> a) *What are the instructional strategies and recommendations for the preperation part of the course?*
>
> b) *What are the instructional strategies and recommendations for the implementation part of the course?*

To answer those questions qualitative data were collected throughout the course under the framework of Design-Based Research.

1.5 Theoretical Framework

It is essential to put forward what theoretical framework is and how it will help to manage a study before continuing with the selected theoretical frameworks and their explanations. Anfara and Mertz (2006) defined the theoretical frameworks basically as “any empirical or quasi-empirical theory of social and/or psychological processes, at a variety of levels (e.g., grand, midrange, explanatory), that can be applied to the understanding of phenomena” (p. xxvii). This study started with the pragmatic stance to be open to any way to go further with the research. As the starting point for the design of the course Merrill’s First Principles of Instruction was chosen.

According to Merrill (2013), different than learning, instruction is a goal-directed activity towards a specified knowledge and skill and the purpose of instruction is to promote effective, efficient, and engaging (e3) learning. Using both appropriate instructional principles that were developed by the instructor and known instructional strategies are useful to develop a better instruction. Merrill’s (2002) first principles of instruction consists of 5 main principles which are: Learning is promoted when… (a) learners are engaged in solving real-world problems; (b) existing knowledge is activated as a foundation for new knowledge; (c) new knowledge is demonstrated to the learner; (d) new knowledge is applied by the learner; (e) new knowledge is integrated into learner’s world. These principles were applied throughout the course as Merrill (2013) suggested as figure 1.1 below.

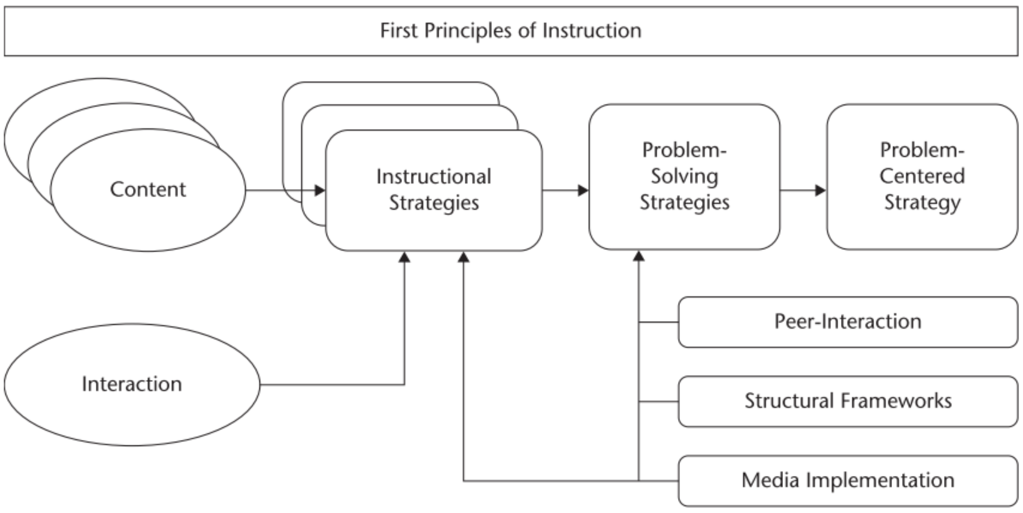


Figure 1.1 First Principles of Instruction - General Process (taken from Merrill, 2013, p. 1)

Instruction would not be effective if not carefully designed, and a unique theoretical framework could be helpful for the design such as First Principles of instruction (Lo & Hew, 2017; Merrill, 2013). The principles used for instructional-design should be design-oriented and relevantly designed to promote learning activities (Merrill, 2013). While designing a problem-centered instruction four phases should be and were taken into consideration for the design of this course (1) activation of prior experience, (2) demonstration of skills, (3) application of skills, and (4) integration of these skills into real-world activities (Merrill, 2013).

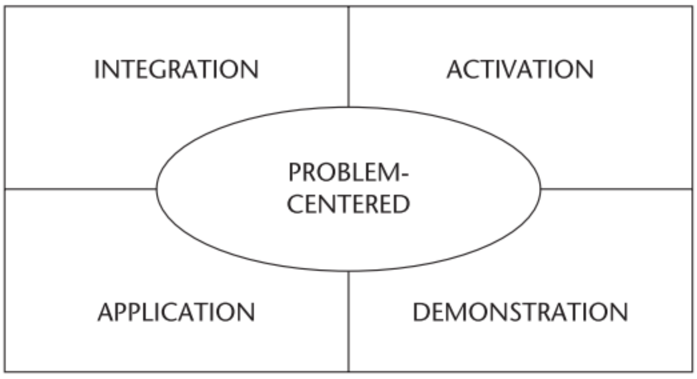


Figure 1.2 Phases of Instruction (Merrill, 2008)

The premise of Merrill's of first principles instruction was providing an efficient, effective, and engaging instruction (Merrill, 2008), similar to the goal of this study. Therefore, it was selected as the main theoretical framework to develop the course from the starting point to the extracting of instructional strategies.

1.6 Assumptions

Following statements assumed to be true for this study:

1) Participants honestly answered the questions of interviews during the study and willingly stated their opinions
2) The data were accurately recorded and analyzed
3) Validity and reliability techniques are sufficient to meet the needs of the study

1.7 Limitations

This study has some limitations in terms of generalizability. Following limitations could affect the generalizability of the study:

1) This study is limited to students in Middle East Technical University.
2) Sampling is another limitation of this study because purposive sampling was used.

1.8 Delimitations

1) This study is delimited to students who (a) are non-programmers or novice programmers, (b) have basic computer skills (c) are students of Middle East Technical University.
2) This study is delimited to data from interviews, observations, documents, products of the students and literature.

1.9 Definitions of Terms

Visual Programming: Visual programming language is "a programming language that lets users to create programs by manipulating program elements graphically rather than specifying them textually" (Smutný, 2011, p. 358).

Computational Thinking: … an analytic approach to problem-solving, understanding human behavior by drawing on the concepts fundamental to computer science (Wing, 2006).

App Inventor: “MIT App Inventor is a drag-and-drop visual programming tool for designing and building fully functional mobile apps for Android” (Pokress & Veiga, 2013, p. 1)

CHAPTER 2

LITERATURE REVIEW

This chapter consists of two main parts. While the first part focuses on programming education and suggestions based on it (including visual programming languages and computational thinking), second part discusses firstly, mobile technologies and importance of mobile application development and secondly, visual programming environment and its features. The main environment that this study focused on is App Inventor (AI), but Scratch is very similar and older program than AI. There are more research studies about Scratch, and it is essential to emphasize its features, so Scratch will also be examined under this part. Since literature is an important source of information especially in design-based research, in both parts, suggestions, and information were used to design the first implementation of the study.

2.1 Computer Programming Education for Novice Learners

Computers and computer-related knowledge are an important link between the world's economy, technology, and innovation (Buitrago Flórez et al., 2017). In recent years, people, especially younger ones, tend to use computers and technology with an increasing rate, however, computer programming is still viewed as a terrifying field (Sandoval-Reyes, Galicia-Galicia, & Gutierrez-Sanchez, 2011). It is a known fact that computer programming is also seen as a subject that is difficult to teach and learn (Cetin, 2013; Ma et al., 2011; Mozelius et al., 2013; Sandoval-Reyes et al., 2011). This is true for not only the K-12 students but also the students in higher education. Some studies state that even the 50% of computer science students switch to a different study program due to high cognitive demand after three or four semesters (Schäfer et al., 2013). Moreover, Kurkovsky (2013) stated that "CS majors end up changing their majors after taking CS I or CS II, because they find the material irrelevant to practical applications" (p. 138). The key to the solution of this problem regarding non-majors must be the design of introductory programming courses. When it comes to the solution, there is no consensus about how an introductory programming course should be designed (Mozelius et al., 2013). This is not just the problem of computer science students/teachers either. According to Guzdial and Forte (2005), many students who are not majors in computer science are going to be required to take an introductory computing course which includes programming as a core activity. For example, Georgia Tech University requires all of the students to take a course in computers science (Porter et al., 2013). However, Porter et al. (2013) also stated that the pass rate of majors of mentioned majors was less than 50% on average. Obviously, there is a problem on learning programming for novice programmers.

New knowledge and technologies (more enjoyable environments, changes in the classroom environment and ways of learning etc.) lead computer science education to evolve, based on those global changes (Hawi, 2010; Malliarakis et al., 2013). Even though there are lots of challenges in computer programming environment, there are also studies based on those changes and technologies provide some approaches, suggestions, and changes in computer programming and its education to solve those problems. Those suggestions and how they can be relevant when designing a new computer programming course will be examined in the following sections.

2.1.1 Approaches, Suggestions, and Shifts in Programming Education

First suggestion for programming education depends on the paradigm shift in education. Knowledge age needs a student-centered educational system (Charles M. Reigeluth, 1999a). A student-centered approach should be used in computer programming learning rather than a traditional one because its aim is not that students memorize or stack information in their minds (Hawi, 2010). It requires higher thinking skills such as problem-solving, creative thinking, to cope with the complex challenges. The teacher should not be the instructor of the course, s/he should be a guide and help students when they need help. Additionally, when the teacher steps back a little, s/he gave students a chance to think and become a source of information (Hawi, 2010).

Sometimes, teachers could bore or get students out of the flow by presenting too complex problems and exercises which are higher level than students' knowledge. Apiola and Tedre (2012) reported that providing simple tasks for an introduction to students will help them to be motivated and make them feel competent for the course.

Reigeluth's (1999a) elaboration theory also suggests that given tasks to students should proceed from simple to complex.

Another suggestion is about the assessment of programming courses. Students should not be choked with the traditional lecture-homework-exam arrangement, on contrary, they should be free to create their own program based on their need (Apiola & Tedre, 2012). Open-Ended assignments should be provided to students which can easily be extended based on students' needs (e.g. students could create a part of their application project) (Falkner & Falkner, 2012). Those programs should be assessed according to criterion-based assessment in which criteria should be stated carefully (Barg et al., 2000). Grading should not be the main source of motivation in programming education. "Success of problem-solving activities based on its secure environment, in which students explore and experiment without worrying about grades" (Hawi, 2010, p. 50). Creating their own program based on their daily or academic needs should be targeted to be students' primary motivator. There is a consensus about that programming learning cannot be effective without the practice of students in their leisure time (Apiola & Tedre, 2012). So it should be aimed to motivate student about working on their own project in their free time. Additionally, assessment should based on the products, not on some kind of test or exam. As Marlowe and Page (2005) stated the importance of learning by doing in programming, they emphasized that best way to check the understanding of programming is programming. Novice programmers must create their own program plans, in this way problem-solving skills could be developed (Robins et al., 2003). The learner should be the driver of his own learning, not a passenger of learning goes along with the classroom.

Apiola and Tedre (2012) subtracted some pedagogical suggestions on the programming teaching: First number of practices should be increased; second, in-class exercises the best ways of utilizing guided environments should be explored; third, while positive aspects of group work is put to use, negative aspects should be overcome; fourth arranged interventions should be included with more action and formative research; and the last, help should be always available for students from different sources.

Falkner and Falkner (2012) stated that for novice learners, learning activities should have a low threshold and high ceiling which means exercises should be easy to complete but also provide students a way that they can be challenged if they want. In Apiola and Tedre's study (2012), teachers suggested there should be simple programming language which can direct learners to an advanced programming language. To create a successful programming course, instructors should avoid designing a course that leads students to complain frequently: Programming is asocial, boring, tedious, irrelevant and competitive (Porter et al., 2013). What they should do is create a social environment (online or real world), motivating student to create their own programs based on their needs, and rather than grades their program should be the main motivation source.

Kafai and Burke (2013) mentions about 3 shifts in programming: First, shift from code to applications in which students create programs rather than coding exercises; second, shift from tools to communities, where tools like scratch and AI provide an intuitive environment, it is also important to create a community as well; third shift from creating from scratch to creating via remix, programming is not an individual activity anymore and students should use samples and sharing as well. Those 3 shifts should be taken into consideration when creating any programming course to keep up with the new programming paradigm.

Saito and Yamauara (2013) stated that traditionally "There are two approaches in learning a programming language: a bottom-up approach (BUA) and a top-down approach (TDA)" (p. 16). In BUA, learner first learns the language syntax and grammar; proceed from basics to details. On the other hand, TDA provides learners a sample program to let learner study on the sample to understand the language and programming. Saito and Yamaura (2013) emphasize that advantages of TDA over BUA it needs a shorter period to understand, students enjoy more from reusing and understanding the code and focuses more on the program rather than syntax. On the other hand, TDA could lead students to "I think I understand everything" syndrome which is understanding the overview of the program but may not understand the detail (Saito & Yamaura, 2013). Sorva (2013) also states that students could get a concept wrong with an incomplete understanding with TDA. However, new visual environments like Scratch or AI could overcome the disadvantages of both approaches. Users can both create their own application without a syntax and understand every step of what they have created. Due to its easy-to-understand nature, those environments could integrate the advantages of both approaches. Even with a BUA, they need shorter period, provides enjoyment, and focus on the program rather than syntax which may motivate students easily. Motivation in programming education will be examined thoroughly in this chapter.

2.1.2 Computational Thinking

Computational thinking is defined as an analytic approach to problem-solving, understanding human behavior by drawing on the concepts fundamental to computer science (Wing, 2006). People use computational thinking without knowing that they use it. It should be learned how to use it and encouraged to be used. Computational thinking can be used while solving a complex task or designing a complex system by reformulation it into little pieces that we know how to solve (Wing, 2006). "Computational thinking is often associated with computer science actually is better understood as extending computer principles to other disciplines in order to help break down elements of any problem, determine the relationships and solve it with devised algorithms" (Kafai & Burke, 2013, p. 62). While industry age was demanding specialized in one job for mass production, information age needs more people with employees with problem-solving and group-work capabilities. This type of thinking skills could help people to cope with modern day problems, additionally, it is a desired quality for business life.

Most of the people still think computational thinking as just about computer science as it is mentioned before. They confuse about what computational thinking is what it is not. Wing (2006) lists the characteristics of computational thinking by mentioning what it is and what it is not, to clarify this confusion as follows:

- Conceptualizing, not programming
- Fundamental, not rote skill
- A way that humans, not computers, think
- Complements and combines mathematical and engineering thinking
- Ideas, not artifacts
- For everyone, everywhere
- Intellectually challenging and engaging scientific problems remain to be understood and solved

Computational thinking is no longer a special thinking type for software developers, computer scientists etc. Wing (2006) also states that "computational thinking is a fundamental skill for everyone, not just computer scientists" (p. 33). It should be encouraged to learn not only for computer majors but also for all K-12 and university students. Although computers are part of our lives nearly 30 years, computational thinking is not still a part of our curriculum (Kafai & Burke, 2013). Some institutions take baby steps to meet the needs. For example, Georgia Tech University requires all of the students including Liberal Arts, Architecture, and Business majors to take a course in computers science (Porter et al., 2013).

Kafai and Burke (2013) stated that while teaching computational thinking, and programming language (syntax, properties of the language) should not be the focus of teaching. An easy to learn programming could help students to focus on computational thinking and problem-solving rather than focusing on syntax (semicolons, parentheses, debugging). It is essential for a teacher, especially in an introductory programming course, provide students a programming-centered learning environment rather than a programming language-centered environment.

Kwon et al. (2011) propose an integrated learning environment in which students learn necessary information about programming language for their program when they need to learn. Instead of the passive use of syntax, programming must be taught in the light of computational thinking (Buitrago Flórez et al., 2017). This minimizes the language learning section (and of course its cognitive load) and focuses more on the programming and computational thinking process. One of the pioneers and supporter of teaching computational thinking is MIT. "MIT Logo Group in the 1960s, and whose influence persists today through many activities and programs designed to support computational thinking" (Castelluccio, 2012, p. 67). MIT Media Lab still supports computational thinking by creating visual programming environments such as Scratch and AI. Visual programming environments could provide an easier way to learn both programming and computational thinking. Visual programming and environments will be examined in the next section.

2.1.3 Visual Programming

From the inventions of programming languages, many tools were created and used to enhance the learning and motivation of novice programmers (Hooshyar, Ahmad, &

Nasir, 2014). Visual programming language is "a programming language that lets users create programs by manipulating program elements graphically rather than specifying them textually" (Smutný, 2011, p. 358). Learning syntax could be one of the barriers that novice programmers face while learning programming. Winslow (1996) states that novice programmers know the syntax and meaning of statements but they do not know how to create a program with this knowledge. Novice programmers sometimes think that programming is the production of program text rather than controlling computer's actions at runtime (Sorva, Lönnberg, et al., 2013).

Sorva (2013) states that one way that could help learners to understand program dynamics is visualization. While most of the teachers use visualization on textbooks and drawing on paper, some of them use visualization software (Sorva, Karavirta, et al., 2013). Sorva, Karavirta, et al. (2013) and Ma et al. (2011) emphasize that visualization software tend to offer a positive impact on introductory programming courses. Sorva, Karavirta, et al. (2013) also state that visual programming languages could be also used for visualization. Users can construct programs by dragging and dropping graphical objects (Falkner & Falkner, 2012).

"Visual Programming could be a good solution to help non-programmers learn programming more easily" (Hsu & Ching, 2013, p. 120). Visual programming languages are easier, and closer to human language since it uses representations, while traditional languages are more powerful in terms of creating complex programs (Lye & Koh, 2014). Visual programming can help novice programmers to avoid syntax errors and make programming a more enjoyable experience (Hsu & Ching, 2013). Focusing on syntax and errors could increase the cognitive load of novice learners. Students frequently forget to write a semicolon or a parenthesis and stuck in the same place without knowing what to do (Bennedsen & Caspersen, 2012). Bennedsen and Caspersen (2012) state that even the students who understand the logic of loops and statements could not remember how to write it. On contrary visual programming let them create their program without getting stuck in the syntax. As in Scratch and AI example, only the specific blocks are relevant to integrate and prevent users from making mistakes. Soloway and Spohrer (1989) also suggested the use of visual programming, especially for the novices. Lye and Koh (2014) reported that visual programming environments can help learners to understand computational concepts more easily without the need to learn syntax. Visual programming languages like Scratch and AI could provide a handy and fun environment for an introductory programming course. In the next part, visual programming environments will be examined.

2.2 Programming environment and Mobile Technologies

In this part of the paper, the main focus is the environment that will be used in the study: App Inventor. AI is a visual programming environment that is designed to create applications for Android OS smartphones. To highlight its importance before mentioning AI, smartphones, Android OS and mobile software implications will also be examined. Additionally, Scratch (another visual programming environment) will be examined because of its similarity to the AI environment.

2.2.1 Mobile Technologies and Android OS

Abelson (2009) states that 10 years ago, people's use of computing was largely dissociated from real life. Now computers are in our pockets (smartphones), our TV (smart TVs), in our handbags (tablets) etc. There is an application for nearly anything to solve daily problems of people or just to entertain them. Especially mobile operating systems changed people's lives dramatically. After the development of Apple iPhone and Google's Android OS phone, a new generation of computers have risen. According to the International Data Corporation (*IDC: Smartphone OS Market Share*, 2017), Android OS is the leader with the 85% market share as of the first quarter of 2017. There are hundreds of thousands of applications in the Android OS' application market, called Google Play Store. Android OS offers free and open source development kit (SDK) for developers. However, developing applications with Android SDK is complicated for novice programmers and it is complicated to teach in class. However, Abelson (2009) had a vision for Android where young people -and everyone- can engage the world of mobile services and applications as creators, not just consumers. Abelson cooperated with Google to develop an easy to develop visual programming environment: AI. With AI, (released in 2010) ordinary folks with no programming experience can write their own applications (Castelluccio, 2012). This environment can help both students and teachers to develop their own applications which are not very easy in normal conditions (Soep, 2011).

Development of mobile apps attract teachers attention, however, applications for every need of teacher is not available sometimes (Hsu & Ching, 2013). Most of the teachers even with little ICT knowledge, can prepare presentations with computer programs, however, application development is a far more complicated job to do. Designing and developing mobile apps is still a challenge for teachers without programming experience (Hsu & Ching, 2013; Tanner & Duncan, 2013). AI could provide an easy-to-use environment for both teachers and students. Teachers and students could design their own applications according to their needs. They can also change/edit the existing projects to reach what they want (Hsu, Rice, & Dawley, 2012). There is a very similar environment to AI was released by the MIT in 2007. Since it is similar to AI, Scratch will be examined first.

2.2.2 Scratch

Scratch is an educational programming environment which lets users create interactive media-rich projects, created by MIT-media lab and Yasmin Kafai's team from UCLA (Kwon et al., 2011; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Scratch project publicly launched in 2007 (Maloney et al., 2010). Novice programmers can learn programming with interest while creating their own games or animations by using Scratch environment (Kwon et al., 2011). While Scratch is created mainly for K-12 students, Scratch has been used for university (including Harvard University and UC Berkeley) students too (Resnick et al., 2009; A. Wilson et al., 2012).

Scratch has a site that includes projects, tutorials, forums etc. Users can get help from other scratch users and examines each other's projects. Core audience of the site is children between 8 and 16, however adults are also participate to the site (Resnick et al., 2009) Resnick et al. (2009) stated that scratch users upload more than 1500 new projects every day, since release more than 3 million projects of 1.5 million registered members (Meerbaum-Salant, Armoni, & Ben-Ari, 2013) have been uploaded which are open source and shared with other users.

While creating their programs with Scratch, they also learn creative and systematic thinking, problem-solving skills, mathematical and computational concepts which are parts of computational thinking (Lee, 2011; Resnick et al., 2009). Scratch provides an easy to design environment in which user can even make changes while a program is running (Resnick et al., 2009). In that way, it also encourages users to experiment.

Scratch has recognized as a highly potential first language for first-time programmers (Tangney, Oldham, Conneely, Barrett, & Lawlor, 2010). Mostly used the first example of programming, “Hello world” is just a two-piece puzzle in the Scratch environment (Malan & Leitner, 2007).

Studies also show that students also find Scratch as a fun to use environment (Malan & Leitner, 2007; Maloney et al., 2010). Scratch environment is coherent with the low threshold (easy to learn), high ceiling (could be created complex projects) concept which is a needed feature for introductory programming languages (Su, Yang, Hwang, Huang, & Tern, 2014). Wolz et al. (2009) reported that after initially learning Scratch, the students’ transition to Java or C appeared to be easier. Scratch has a very similar interface and logic with AI. However, Scratch is designed and developed for children at its core and products of it are animations and games. On the other hand, users can develop a standalone application for a smartphone or tablet with Android OS. Next section will focus on Android and its features.

2.2.3 App Inventor

“MIT App Inventor is a drag-and-drop visual programming tool for designing and building fully functional mobile apps for Android” (Pokress & Veiga, 2013, p. 1). Google App Inventor for Android was created for users without coding experience to make simple apps for a mobile phone which was released in 2010 (Bertea, 2011). By the September 2017, App Inventor has 6.8 million registered users from 195 countries and in total, 24 million applications were developed by the users (MIT App Inventor, 2017).

In traditional programming learning, it took months to create a program that actually works rather than a console application like ordering inputted numbers or generating a list of prime numbers. It only took15 minutes in AI to create sound box application which is very popular in Google Play Store. Additionally, you can send and install it to your phone as soon as it is finished. This could increase the motivation of students toward learning programming. AI has two main component: designer part and blocks editor part (Pokress & Veiga, 2013). While blocks editor is used for the behavior of the application (programming part), designer let the user design and place the components to the screen. A screenshot of the designer screen can be seen below (figure 2.1). Components selected and dragged to screen to add to the appliaction. The components added was listed in the component section. The properties of each component can be changed from the properties section. In designer screen, there are many components to use, including buttons, textboxes, GPS, Bluetooth detectors etc. User can just drag those components to the screen and add behaviors to them in the blocks editor (Bertea, 2011).

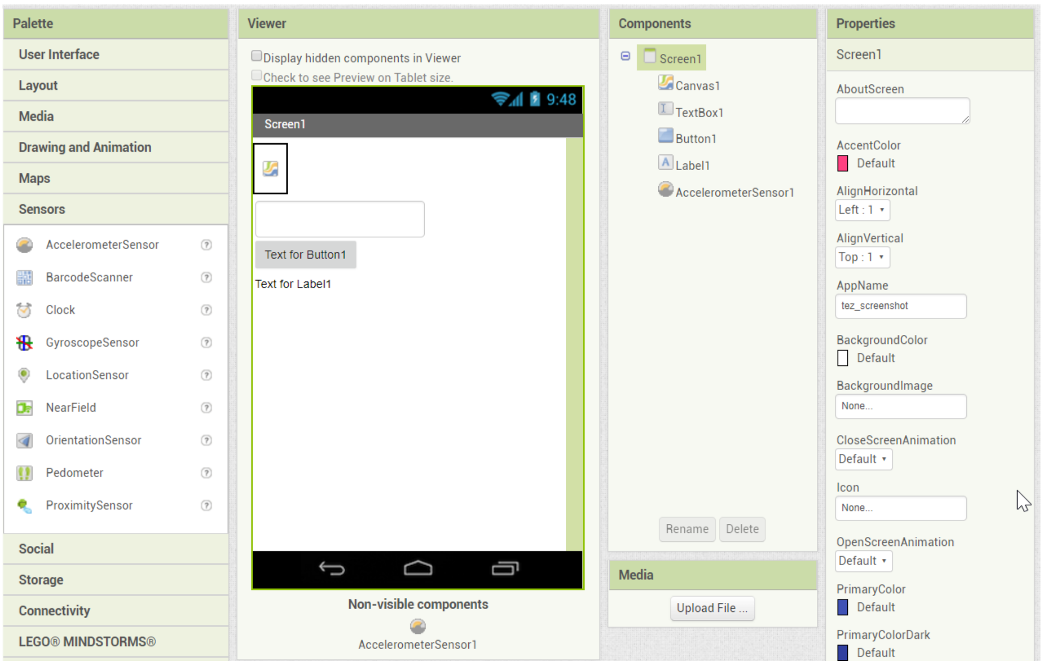
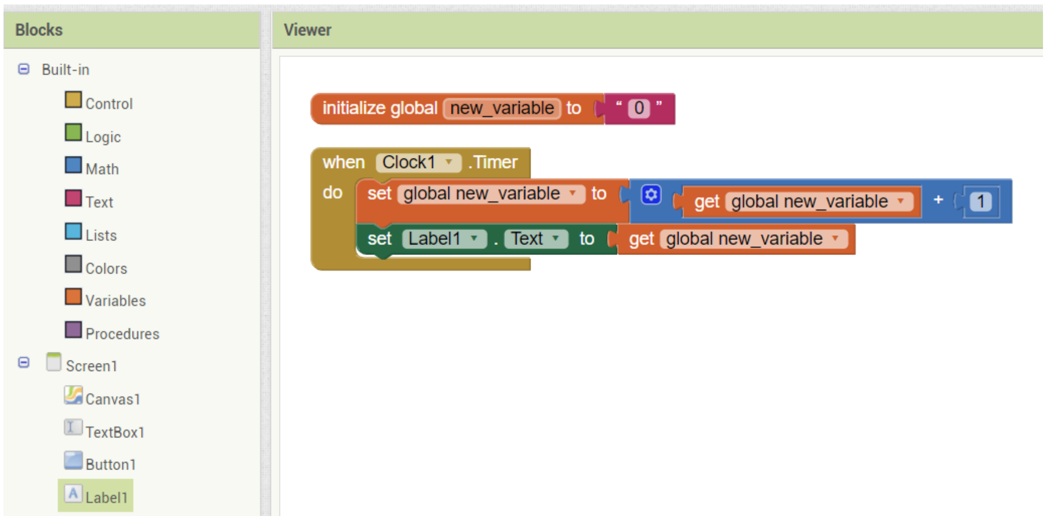


Figure 2.1 Designer screen of App Inventor

AI focuses on the functionality of the application. Users can create their applications intuitively without any programming knowledge just by exploring the components (Bertea, 2011; Pokress & Veiga, 2013). Visual cues that AI provides, reduces the chance of errors (Sandoval-Reyes et al., 2011). AI makes application development considerably easier than a traditional programming language (Smutný, 2011). It uses puzzle shaped blocks to help students program their application. As it can be seen from the figure 2.2 below, programming part was made with the color categorized blocks. Users create their applications by dragging and dropping the blocks to the viewer screen, and connect or nest them into the compatible blocks.



```
initialize global new_variable to " 0 "

when Clock1 .Timer
do  set global new_variable to  get global new_variable + 1
    set Label1 . Text to  get global new_variable
```

Figure 2.2 Block-based programming interface of App Inventor

Smutný (2011) lists key features of the visual language of AI as

- has no syntax,
- based on idea what happens when component do a certain action
- No need for a manual - Drag and Drop
- Restrict users from making mistakes
- Concrete
- Has a powerful library

While the primary audience of App-Inventor is educators and learners, it is also used by developers, entrepreneurs, and hobbyists (Pokress & Veiga, 2013). Hsu and Ching (2013) stated that AI has a great potential for novice programmers to develop apps even for professional needs.

2.2.4 Advantages of App Inventor

The first advantage of the AI is that it provides a powerful medium to teach computational thinking to students. AI could also provide an easy-to-use environment to let teachers use for presenting introductory programming concepts. Hsu and Ching (2013) stated that AI motivates students to learn programming logic and to engage in the creative problem-solving process. As it was stated before, in traditional programming environments, students could focus too much on the syntax and less on the algorithm and logic behind the program. AI could remove this barrier with its no-syntax environment.

Another advantage of App-Inventor is easy-to-test nature for applications. Users can see what they create concurrently without any need for compiling and running process which encourage users to develop their apps (Pokress & Veiga, 2013). Additionally, users can see their applications running both from the emulator (design screen) and their own smartphone or tablet by using AI Companion app at the same time. Applications can be tested, used or played immediately on a mobile device or emulator (Hsu & Ching, 2013). Users also can test their applications by right-clicking to individual blocks and clicking to "Do it" command to see blocks' behavior (Pokress & Veiga, 2013).

It is also applicable for teaching in higher education. In Hsu and Ching's (2013) study, participants define designing an app with AI as fun and useful. They want to use the program they have created in the class. In the same study, even the participants with programming experience find AI satisfying and incorporate it with Java language. Pokress and Veiga (2013, p. 2) states that college and high-school faculty have successfully used AI in their courses over the 4 years.

2.2.5 Disadvantages of App-Inventor

As the advantages, AI has also some disadvantages. One disadvantage of AI is the fact that, while simple applications are easy to make, complex applications could need deeper knowledge (Bertea, 2011). Additionally, when programs get complicated, block designer gets annoying as well (Hsu & Ching, 2013). If there was an option to switch from blocks editor to text editor, it would be more helpful both for control the program and to be a step for advanced programming.

Additionally, students may not appreciate the tool that their instructor has provided (Sorva, Lönnberg, et al., 2013). Based on the teacher perspective, AI could be seen as a perfect environment for teaching programming, but students may found it childish or meaningless. Of course, this disadvantage could not be detected without a pilot study.

2.3 Motivation/Engagement

Motivation is what makes students continue learning by engaging in learning activities (Boyer, Phillips, Wallis, Vouk, & Lester, 2009). Introductory programming course has a low retention, and high dropout/failure rates, hence the students may have lower motivation levels (Howles, 2009; Mendes, Paquete, Cardoso, & Gomes, 2012; Schäfer et al., 2013). Two probable reasons for that are lack of motivation and fear from programming of students. However, maintaining motivation and self-efficacy, especially in introductory programming courses, do not have enough attention from the instructors and organizations (Parhami, 2008; Soh, Samal, & Nugent, 2007). Additionally, it is hard to constantly maintain engagement in programming education because failure could lead to confusion and frustration (Bosch & D'Mello, 2015). As it can be seen in the affect transitions model figure below (figure 2.1), while curiosity could trigger the engagement, confusion could lead to frustration, and frustration could lead to disengagement. Those factors should be taken into consideration while designing the course.

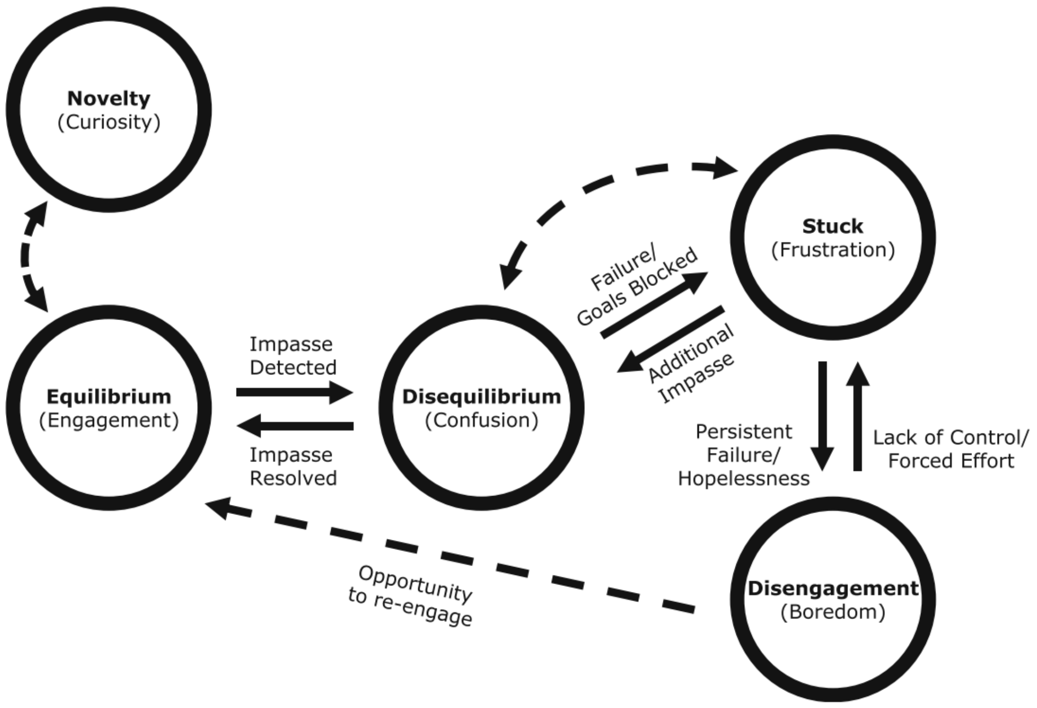


Figure 2.3 Affect Transitions Model (taken from Bosch and D'Mello 2005, p. 201)

The experience in the last 40 years shows that learning programming is hard and introductory programming courses have a high dropout or failure rate (Bati, Gelderblom, & van Biljon, 2014; de Raadt, 2007). Students with low computer experience are likely to be more anxious when starting programming course (Byrne & Lyons, 2001). Sandoval-Reyes et al. (2011) stated that "motivating students to learn programming has never been easy" (p. 444). It is also a known fact both from research studies (Mozelius et al., 2013) and personal experiences of the author of this study, programming courses which are mostly teacher-centered and syntax-focused are found boring by the most of the students. Instructors should be very careful while designing a programming course for novice learners because they are likely to be anxious and negative about the course.

Various educational methods have been proposed over the years to increase student interest and motivation towards programming and computer science (Kim, Kim, & Kim, 2013). Hawi (2010) "emphasizes that the social dialogues among students in class generate better communication outside the class" (p. 52), which will motivate students and should encourage them both in class and online discussion groups. Another motivation source should be the environment. Visual programming languages could provide an environment which is easy to understand and productive in the end. In that way, students find it enjoying and attractive. Robins et al. (2003) also state that working on easy tasks especially with graphical output can be stimulating and motivating.

The most important motivation type in learning is intrinsic motivation. Long's (2007) study shows the factors that influence intrinsic motivation in programming education (*figure 2.1*). All of the subtopics are about being able to actually do something rather than writing "Hello World" on console screen. Students should have the chance to create their own working programs to be motivated about programming.

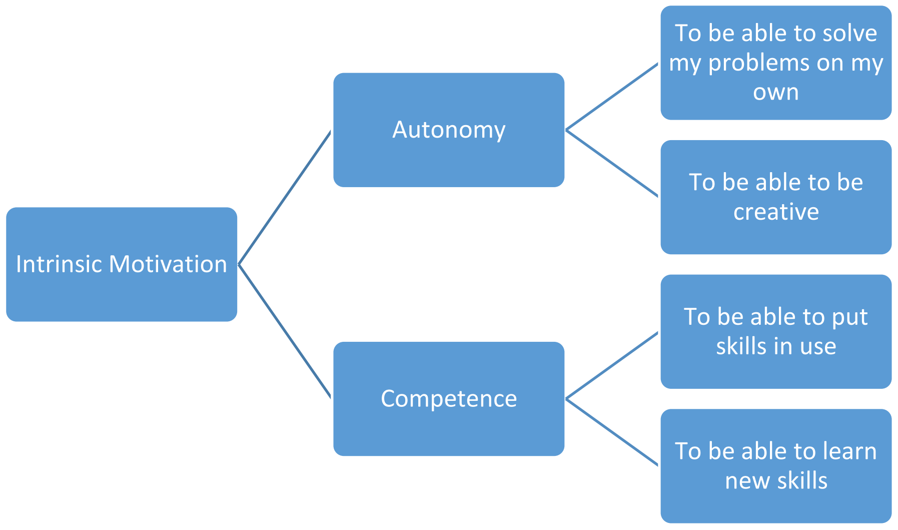

Intrinsic Motivation
Autonomy
Competence
To be able to solve my problems on my own
To be able to be creative
To be able to put skills in use
To be able to learn new skills

Figure 2.4 Factors that influence intrinsic motivation in programming learning (Long, 2007)

It should not be forgotten that many students make very little progress in a first programming course (Robins et al., 2003). The main aim should not be to teach everything in one semester or it could demotivate student to learn programming. Teachers should mainly focus on teaching algorithmic and computational thinking to in a motivating environment. Robins et al. (2003) suggest that instructors "…need to motivate students, engage them in the process, and make them want to learn to be effective programmers" (p. 166). The instructors should put the motivation in the first place, even before the content of the course, for the first time programming learners.

2.3.1 Ensuring the 3C (Confidence, Competence, Comfort)

A large number of studies showed that students think that programming is difficult to learn, and teachers also think that it is difficult to teach (Lister et al., 2004; Robins et al., 2003; Wong, 2017; Yadav, Gretter, Hambrusch, & Sands, 2017). "Engaging students is critical too deeper learning" (Guzdial & Soloway, 2002, p. 18). Making sure of student feel confident, competent, and comfortable should be one of the engagement strategies that instructor implements. Students feel more motivated when they have a higher level of self-efficacy and competence beliefs (Pintrich, 2003). Self-confidence is one of the factors that influence motivation, and student motivation is an important, directly affecting element of computer science learning (Boyer et al., 2009; Cheong, Pajares, & Oberman, 2004). Wiedenbeck's (2005) study also shows that self-efficacy is one of the key factors that affects the outcome of non-majors' programming learning. Bergin and Reilly (2005) also reported that self-confidence is strongly correlated with programming performance. Additionally, students with low self-efficacy are unlikely to seek instrumental help and ask more unnecessary questions (Cheong et al., 2004). Katz, Allbritton, Aronis, Wilson, and Soffa (2006) reported that loss of confidence and decrease of interest might be the cause behind the dropout problem. Schunk's (1981) study showed that there is a high correlation between self-efficacy and achievement. Wiedenbeck (2005) also stated that having more self-efficacy helped students to get through challenging tasks. Therefore, making students feel competent and confident throughout the course is important for ensuring the motivation of the students and reducing the dropout rate. Soh et al. (2007) suggested adding methods to the curriculum to focus on and measure the self-efficacy and motivation. Self-efficacy of the students get higher throughout the course according to the observations of the instructor and the interviews conducted at the end of each semester. Wiedenbeck's (2005) study also emphasized that post-self-efficacy had a high correlation with performance.

Robins et al. (2003) stated that according to more than one studies, students reveal how they are likely to do in the first two weeks of the programming course and many of them make very little progress in a first programming course. "The exploratory findings suggest that student motivation is an important component of the process by which students come to understand computing." (Boyer et al., 2009, p. 114). Wong's (2017) study put forward that students are afraid of programming because it is too complex for them. Lack of performance affects confidence, study habits, engagement, retention (Falkner & Falkner, 2012). It is important to keep the motivation of the students high in the beginning of the course to help them feel confident. Gasparinatou and Grigoriadou (2011) reported that learners with appropriate knowledge tend to take the path of least resistance. Therefore, as Reigeluth (1999b) suggested for any instructional design, tasks should be provided simple to complex. Especially initial course material should be simple and they should be expanded as students being more experienced (Robins et al., 2003). In addition to the simple to complex tasks, there should be an artifact on the hand of the student at the end of each task. In this way, students could say "I know programming" which is also relevant to the last two steps of Keller's (1987) ARCS model: Confidence and Satisfaction. Wilson and Shrock's (2001) study showed that comfort level which is determined by the anxiety level of students while working on assignments, is the best predictor of success in an introductory computer science course. Constituting a comfortable and relaxed environment regarding classroom, communication medium and communication itself is directly in relation with the engagement level of the student. When students are more engaged and motivated to the course, they would be more successful as the studies above suggested.

2.3.2 Communication

Peer help with the additional sources can be effective throughout the course (Soep, 2011). Students' asking for help from other students is their first option (Araujo, Bittencourt, & Santos, 2018). Student control during the course hours may be an important motivator for the students (Boyer et al., 2009). Students who are less comfortable socially may avoid help-seeking (Cheong et al., 2004), however, using the communication medium as the main communication hub could help those students to communicate with their peers. Sanders (2002) stated that the most common problem among students was finding time to meet and work together. Use of communication medium could overcome the time-consuming part of working together. Positive feedback with correction during the course and for assignments submitted to the communication medium increases student's self-efficacy which is directly related to success (Boyer et al., 2009). Using communities are helpful to build a sense of belonging, to develop a sense of self-confidence, and to improve learning (Howles, 2009). Using a communication medium as the hub of the community was decided in this study, even though it has some flaws.

There are two different types of help-seeking: executive help-seeking which is requesting from someone else to complete the task and instrumental help-seeking which is requesting necessary assistance to complete a task (Cheong et al., 2004). Cheong et al. (2004) stated that while executive help-seeking is not a desired behavior, instrumental help-seeking is a useful and adaptive behavior. Instrumental help-seeking should be encouraged through the communication medium and in-class collaboration.

Communication with instructor is important for introductory programming courses as well as others (Howles, 2009). Cooperation and communication with their professor lead students to have greater motivation to learn and better results in learning. "Student must construct knowledge assisted by guidance from the instructor and feedback from other students." (Buitrago Flórez et al., 2017, p. 843). Additionally, communication among students during the course and out of the classroom should be allowed at all times (Williams, Wiebe, Yang, Ferzli, & Miller, 2002). Collaboration activities like pair programming, group code writing during the course hours could promote learning (Bati et al., 2014). Creating a closer relationship between students and instructors through a sense of community could lead to enthusiastic engagement and team spirit (Bati et al., 2014; Bryson & Hand, 2007). Wilson reported (2004) that communication between instructor and students increases both the success and the motivation. Moreover, Howles (2009) reported that it is the communication that affected satisfaction positively not the change of instructor himself.

2.4 Gap in the Literature

Robins et al. (2003) reported that "Majority of studies about programming education focus on program comprehension, often in experts, and typically based on experimental studies" (p.162). There is little research regarding beginner level programming students who are non-CS majors and by collecting in-depth qualitative data from students. Only qualitative research studies could reveal the topics like communication among students, the friendship of the students, and the effects of the emotional side of the participants as a whole (Cheong et al., 2004). There are not many studies or practices that asking students for course improvement suggestions without providing them predefined choices (Sampaio & Sampaio, 2012). This study aimed to create a better course with students throughout the semester. Instead of comparing environments, or methods as most of the studies in the literature put forward, this study focused on exploring the possible improvement with an in-depth and holistic approach.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter will explain and clarify the methodology of this study. This chapter consists of the research question, research design, description of the setting and the course, participants of the study and sampling, researcher's role and potential ethical issues, data collection, data analysis, and trustworthiness of the study sections.

3.2 Research Questions

Purpose of this study is to extract instructional strategies, recommendations, and experiences from and for an introductory visual programming course (named as Visual Programming with App Inventor). Review of literature showed that programming courses are seen as difficult, abstract and complex by the students. This study aims to create a course that copes with those challenges with the help of instructional technology. Main research question of this study and the sub-questions of the research question are:

> *What are the instructional strategies and recommendations to develop an efficient, effective and engaging introductory programming course for non-CS majors?*
>
> c) *What are the instructional strategies and recommendations for the preperation part of the course?*
>
> d) *What are the instructional strategies and recommendations for the implementation part of the course?*

To answer those questions, interviews were conducted with the students; progress of the students was observed throughout the course, and posts and discussions of the students from Facebook group of the course were also examined. Additionally, relevancy of the findings was checked with the literature.

3.3 Research Design

This study aimed to extract instructional strategies, recommendations, and experiences for an introductory programming course. An introductory programming course was designed and reshaped after interviews and observations. Investigating research methodologies and designs let the researcher see one specific framework the best fitted in such a study. This study was conducted under the Design-Based Research framework. Design-Based Research itself is not a methodology, it can be called a research approach or framework (Herrington, Mckenney, Reeves, & Oliver, 2007). Under this design framework, 3 different qualitative data collection types were used: Interviews, observations, and documents (Facebook group discussions and posts). The qualitative research methodology was followed during the data collection and data analysis. Under the following part, Design-Based Research and its implementation in this study will be examined.

3.3.1 Design-Based Research

First of all, it has to be clarified that what Design-Based Research is and why it was used in this study. Since the use of qualitative research methodologies were essential for the research design, and rationale for the qualitative research will also be investigated afterward.

3.3.1.1 Definition of Design-Based Research

Definition of the design-based research approach is now beginning to clarify but also differentiate especially in terms of naming it (Van den Akker, Gravemeijer, McKenney, & Nieveen, 2006). While the most common name is Design-Based Research to describe this type of research, it has many names in the literature. McKenney and Reeves (2012) listed those names as design-based research, development research, design experiments, formative research, and educational design research. There are also names in the literature such as developmental research, design studies, formative research/evaluation, engineering research (J. Van den Akker et al., 2006). Author of this dissertation adopted the term design-based research because as McKenney and Reeves (2012) stated that (even though they prefer educational design research) it is the most common name and it represents the design of the study adequately.

Barab and Squire (2004) define Design-based research as “not so much an approach as it is a series of approaches, with the intent of producing new theories, artifacts, and practices that account for and potentially impact learning and teaching in naturalistic settings.” (p. 2). Richey and Klein (2007) also defined Design-Based Research (DBR) (or design and development research as they call it) as “systematic design, development and evaluation processes with the aim of establishing an empirical basis for the creation of instructional and non-instructional products and tools and new or enhanced models that govern their development” (p. 1). Similarly, McKenney and Reeves (2012) defined the DBR as a research framework that provides an iterative solution for developing a structure and to solve complex real-world problems.

McKenney and Reeves (2012) compiled attributes of DBR from literature as “adaptive, collaborative, contextual, flexible, goal oriented, grounded, integrative, interactive, interventionist, iterative, methodologically inclusive, multilevel, pragmatic, process-focused, theoretical, transformative and utility-oriented.” (p. 13). While all of them are accurate for DBR, there are more universal features for the process of DBR. McKenney and Reeves (2012) listed the main features of DBR as theoretically oriented, interventionist, collaborative, responsively grounded, and iterative. Similarly Van den Akker et al. (J. Van den Akker et al., 2006) lists the characteristics of DBR as (1) Interventionist: designing intervention in the real world, (2) Iterative: cyclic approach, (3) Process-oriented: focus is on understanding and improving interventions, (4) Utility oriented: measured by its practicality in real contexts, (5) Theory oriented: based upon theoretical propositions. While those are the attributes and characteristics of DBR, the next section describes why DBR framework was selected for this study.

3.3.1.2 Rationale for Design-Based Research

Learning environments are getting more and more complex with the emerging new technologies and paradigm shifts. Influence of research in education to the classrooms and practices was loose and indirect (Walker, 2006). Our field needs a different, novel methodology with a new epistemology to cope with the challenge, in which DBR seems like the strongest candidate (Dai, 2012; Dede, 2005). Reeves (2006) reported that instead of more media comparison studies, there is a need for a better approach in educational technology research.

Richey (1997) stated that in instructional design and technology (IDT) field, practice and research are not sufficiently informed by each other. They are rather disconnected. Design-Based Research could be an important methodology to fill the gap between practice and research. According to McKenney and Reeves (2012), the uniqueness of DBR comes from providing theoretical insights and practical solutions at the same time, not in a laboratory environment but in the real world/natural environment. While DBR provides a unique research design to fill the gap, unfortunately, there is not enough amount of this type of research (Richey & Klein, 2007). Purpose of this study aims to extract instructional strategies and recommendations to create an introductory programming course through the practical implications in an educational environment. Through the procedure, the study intended to develop a course, reshape and construct a better one from its predecessor. DBR process is an iterative, formative, progressive and flexible process (Dai, 2012; McKenney & Reeves, 2012). Decker (2006) stated that DBR could help to make the connection between features of educational innovation and learning principles. "DBR in education concerned with building, testing, modifying and disseminating new practices and artifacts for particular educational purposes" (Dai, 2012, p. 13). This study is coherent with DBR framework's nature because it was designed based on the data from a real course, the new course was shaped on the data, and iteratively implementations were tested and shaped again. It can be seen that the aim of the study and the characteristics of the research design congruent with each other.

Sources of DBR come from three main areas:

- Actual workplace settings and projects.
- Technology (especially novel and innovative ones)
- Theoretical questions prompted by current research and development literature (Richey & Klein, 2007, p. 16)

Accordingly, the research problem of this study emerged from both three categories. Author of this dissertation lectured lab courses including programming for 4 years before this study, development of App Inventor environment which provided a new perspective to programming education, and finally literature support that there is a need for an effective and motivation introductory programming course for non-programmers which could help lowering drop out rate for a programming course. Additionally, the purpose of this study was to extract suggestions and instructional strategies. DBR could provide both theoretical and practical contributions to the field. DBR contribute to building blocks of theory and ultimately leads to theories (McKenney & Reeves, 2012). After investigating other methodologies, as the most relevant methodology, DBR was selected. For such a complex task, an iterative study was needed to validate the outcome of this study. DBR is more of a framework than a single methodology that allows the researcher to use a variety of methodologies. Different types of qualitative data were collected and used to triangulate each other. Based on those grounds, DBR was suitable for this study.

There are alternative methodologies similar to DBR and applicable to this theory which were action research, and grounded theory. Both will be examined to clarify the rationale of DBR in this study. Action research is a methodology “conducted by one or more individuals or groups for the purpose of solving a problem or obtaining information in order to inform local practice” (Fraenkel, Wallen, & Hyun, 2012, p. 589). Research questions of action research are different than traditional methodological approaches (Berg, 2001) which is relevant to this study. “Action research aims at solving specific problems within a program, organization, or community” (Patton, 2002, p. 221). While action research is a viable alternative to DBR, this study did not aimed to solve a specific problem. It rather focused on understanding what problem is, and implementation of a new and evolving course which could help the instructors to design their introductory programming course. On the other hand, grounded theory was another suitable alternative methodology for this study. Grounded theory is a methodology for building theory from data or as a more generic sense to denote theoretical constructs derived from qualitative analysis of data (Corbin & Strauss, 2008, p. 1). Similar to this study, in grounded theory, the theories (or principles) are not generated before a study begins, but are formed inductively from the data that are collected during the study itself (Fraenkel et al., 2012, p. 433). However, it needs a strict constant comparative method to compare the data from research site with other sites and theoretical knowledge. It also focuses on the process of creating a theory rather than paricular theoretical content (Patton, 2002). The theory needs to be generated at the end of study which could take more time than intended.

For this study, what will emerge was not expected at the start of the study, it rather constructed in time. Therefore, even though both the action research and grounded theory was suitable to this study as the research method, DBR was chosen.

3.3.1.3 Methodologies in DBR

DBR uses a variety of different data collection methods (both qualitative and quantitative but mostly qualitative), not limited to, but including case studies, interviews, document reviews, observations, surveys, content analysis, expert reviews, experiments etc. (Dai, 2012; McKenney & Reeves, 2012; Richey & Klein, 2007). Common methods used in DBR are presented in Table 3.1 (taken from Richey & Klein, 2007, p. 40). As it can be seen from the table and literature of DBR studies rely on qualitative methods more than quantitative methods (Dai, 2012; McKenney & Reeves, 2012; Richey & Klein, 2007). Data collection in DBR is usually small and purposive samples, it rather focuses on exploring the theoretical propositions in their own context than generalization from sample to population (J. Van den Akker, 1999). This study also consists of pure qualitative methodology due to its exploratory nature. Richey and Klein (2007) explained why qualitative methodologies preferred over quantitative ones in DBR as a) DBR resists the over-controlled nature of quantitative methodologies, b) DBR studies demands exploratory techniques which can be easier to explain with qualitative methodologies. So it can be said that DBR embraces qualitative over quantitative methodology.

Table 3.1 *Commonly used methodologies used under the DBR framework (Richey & Klein, 2007)*

Type of Research	Project Emphasis	Research Methods Employed
Product & Tool Research	Comprehensive Design & Development Projects	Case Study, Content Analysis, Evaluation, Field Observation, In-Depth Interview
Product & Tool Research	Phases of Design & Development	Case Study, Content Analysis, Expert Review, Field Observation, In-Depth Interview, Survey

Table 3.1 cont'd

Product & Tool Research	Tool Development & Use	Evaluation, Expert Review, In-Depth Interview, Survey
Model Research	Model Development	Case Study, Delphi, In-Depth Interview, Literature Review, Survey, Think-Aloud Methods
Model Research	Model Validation	Experimental, Expert Review, In-Depth Interview
Model Research	Model Use	Case Study, Content Analysis, Field Observation, In-Depth Interview, Survey, Think-Aloud Methods

3.3.1.4 Model Development Research and Plan of This Study

According to Richey and Klein's (2007) categorization, DBR consists of two large categories: Product and tool research and model research. The first category includes documenting the entire development and design process of a tool or product. On the other hand, model research ultimately aims to generate a new model or theory by the light of the new knowledge acquired from practice (Richey & Klein, 2007). The process and relationship between practices and DBR were summarized by Walker (2006) as seen in figure 3.1 below.

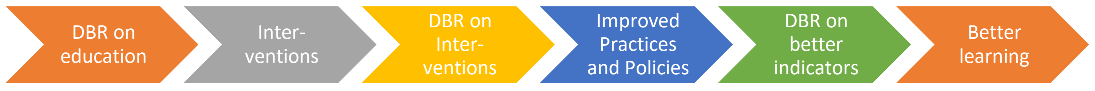


Figure 3.1 Process of DBR and Learning (adapted from Walker, 2006, p. 10)

DBR had a priority on information richness and efficiency that it should not only concentrate on locating problems and shortcomings, but also generate suggestions on how to improve those shortcomings (J. Van den Akker, 1999). Richey and Klein (2007) also suggest that models development research does not mean to focus on the entire process for newly developed models. It could focus on a specific part of the development and/or design process. This study focuses on developing instructional strategies and suggestions for an introductory programming course to students in the university level. This study embraced an exploratory research design. As Richey and Klein (2007, p. 41) stated that exploratory research relates to topics about very little is known and there are few guidelines to follow. Through the explorative and iterative nature, guidelines and procedures emerged. DBR studies in nature reshape again and again after each implementation, due to the nature of the methodology (McKenney & Reeves, 2012).

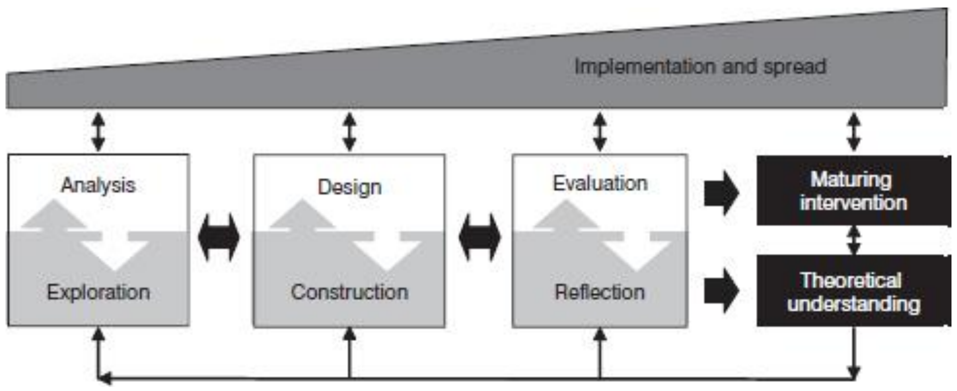


Figure 3.2 Iterative Nature of DBR (McKenney & Reeves, 2012, p. 77)

In the instructional technology and education field, predictive research is widely used. However, it does not meet the needs of educational problems. While predictive research which dominates educational research field for decades (Herrington et al., 2007) approaches propose a well-structured system for educational research, DBR in education proposes an open and an ill-structured system rather than well-structured (Dai, 2012). In DBR, knowledge is constructed during research (McKenney & Reeves, 2012). Figure 3.3 (Reeves, 2006) shows the steps of predictive research and DBR. Reigeluth (1999b) states that instructional-design theories should be prescriptive rather than predictive. DBR is also relevant with prescriptive nature. The similarity of design research and prescriptive theories could be seen from the figure 3.3. Instructional strategies that emerged from this study, were also aimed to be prescriptive. It aimed to guide teachers with principles to create their own introductory programming courses.

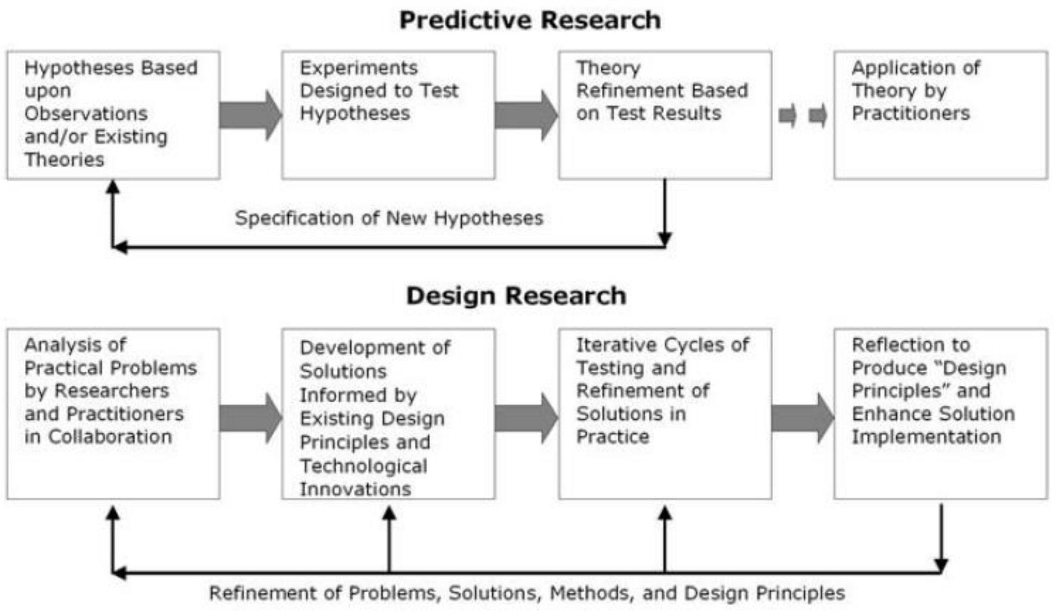

Predictive Research
Hypotheses Based upon Observations and/or Existing Theories
Experiments Designed to Test Hypotheses
Theory Refinement Based on Test Results
Application of Theory by Practitioners
Specification of New Hypotheses
Design Research
Analysis of Practical Problems by Researchers and Practitioners in Collaboration
Development of Solutions Informed by Existing Design Principles and Technological Innovations
Iterative Cycles of Testing and Refinement of Solutions in Practice
Reflection to Produce "Design Principles" and Enhance Solution Implementation
Refinement of Problems, Solutions, Methods, and Design Principles

Figure 3.3 Steps of predictive and DBR (taken from Reeves, 2006, p. 59)

This study also followed the steps of Reeves (2006) suggested. General steps of this study summarized visually in Figure 3.4. As the first main step of the study implemented in spring semester 2014. After the first main (or macro) iteration, iterative cycles continued and the course was opened for a second macro iteration on fall 2014. Details of the courses and cycles will be presented and explained in the following parts of methodoloy chapter.

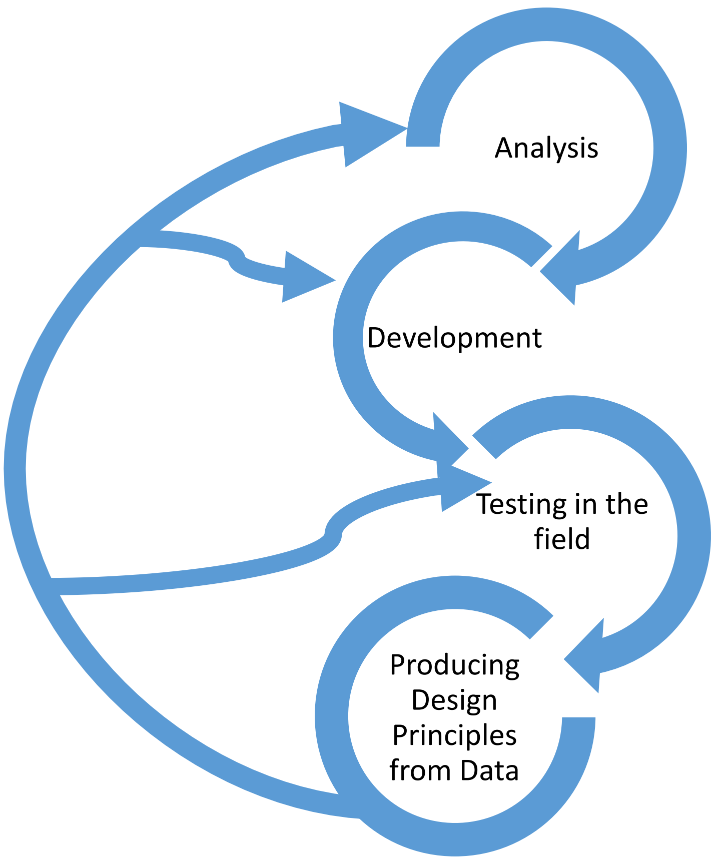


Figure 3.4 General steps of the study

To be more specific, the first design of the study based on the experiences of researcher and literature support. Richey and Klein (2007) suggested that while developing a model (or a theory) a dummy model could be developed before implementation. Merrill (2002) stated that first principles of instruction theory emerged from various theories and applicable to most of the learning environments. Gardner (2010) stated that first principles of instruction is “synthesized through a lifetime of research, practice, and synthesis” (p. 23). First principles of instruction theory was used for the first design of the study as a starting guide. Merrill (2002) lists the five first principles as: “Learning is promoted when; a) learners are engaged in solving real-world problems, b) existing knowledge is demonstrated to the learner, c) new knowledge is demonstrated to the learner, d) new knowledge is applied by the learner, e) new knowledge is integrated into the learner’s world” (p. 43). Those general principals were applied to the first version of the design.

3.3.2 Qualitative Research

Under the design-based research framework data collected and analyzed according to qualitative research methodology. Qualitative research has an important role in this study. Creswell (2007) explained that it is best to use a qualitative study when the researcher aims to reach to an insider view, have a deeper understanding and clear view of the picture. Patton (1985) emphasized that qualitative research aims to “understand situations in their uniqueness as part of a particular context and the interactions” (cited in Merriam, 2009, p. 14). Creswell (2007) mentions that qualitative research has an emergent design per se, which could lead to a change of plans during the implementation of the study. Researcher planned a flexible setting to design the course and gather the data to use this characteristic of qualitative research beneficially.

3.3.2.1 Rationale for Qualitative Research

This study based on a pragmatic and interpretive research paradigm rather than a positivist one, because rather than having one and only answer to the question or having a hypothesis to test, it focuses on building the possible answers through observations and multiple opinions. Qualitative research uses data for exploring and building the knowledge inductively in their natural setting, instead of testing a hypothesis deductively (Creswell, 2014; Merriam, 2009).

As Merriam (2009) explained that interpretive research “assumes that reality is socially constructed, that is, there is no single, observable reality. Rather, there are multiple realities, or interpretations, of a single event. Researchers do not ‘find’ knowledge, they construct it.” (p. 8). As the interpretive paradigm suggested research study should take place in their natural setting to get a clear understanding. Qualitative researchers tend to gather data from thir natural settings, because it gives them the real information they want to observe (Creswell, 2014). Merriam (2009) explained the difference of naturalistic inquiry as taking place in a real-world setting rather than a laboratory, and whatever was being observed and studied was allowed to happen ‘naturally’ in which researcher interested in understanding the phenomena from the people who have experienced and extract the essence of it.

One of the most mentioned strengths of qualitative research is having a deeper understanding of more complex and exploratory research studies. This study sought to observe an introductory class and design it through the needs of the students. Both feedback from the students during the semester and their opinions after the course ended were taken into examination to extract the important points for an introductory programming course. Due to the exploratory nature of this study, qualitative research was selected under the design-based research as a framework. Qualitative research could be effective for both contributing to knowledge of a field and to improve the practice of discipline (Merriam, 2009). Merriam (2009) also stated that the exploratory nature of qualitative research is relevant to improve a practice through exploration and discovery.

The mostly mentioned weak sides of qualitative research are lack of causality and representing the population (Merriam, 2009). However, Merriam and also (2009, p. 5) express some of its strengths as uncovering the meaning of the phenomenon, understanding how people interpret their experiences, how they construct their worlds, and what meaning they attribute to their experiences. Similarly, Patton (2002) stated that main strength of qualitative research is seeing from the eyes of the participant in their natural setting.

Qualitative research and data were the most relevant research methodology and data type for this research study. Merriam (1998) lists the characteristics of qualitative research as, [1] getting the insider view from participants, [2] human is the primary instrument, [3] gathering data from its natural setting, [4] employing inductive research strategy, [5] product is richly descriptive. All of the characteristics represent the distinctive pertinence. Especially its exploratory, inductive nature helped to uncover the unknown parts, and make explicit the tacit knowledge about programming education. Moreover, building the knowledge through the eyes of both participants and the researcher makes it more suitable for this study.

3.4 Description of the Setting and the Course

The course offered as an elective course all across the university. The name of the course in registration system of the university was Special Problems in Computer Education and Instructional Technology. The course was announced throughout the campus with a poster as seen in figure 3.5. In the poster, information to register the course was located as well as the first meeting place and date. The unofficial name of the course was “Visual Programming for Android”.

Android
CEIT440
APP INVENTOR
ile
Uygulama
Geliştirme
Programlama bilgisine ihtiyaç duymadan, akıllı telefonlar için uygulama geliştirmek ister misiniz?
Ders Kodu: 4300440 Section 1
First Meeting: 21 Şubat 9:00, CEIT EFC111
İletişim: kykaya@metu.edu.tr
ODTÜ METU

Figure 3.5 Poster to Announce the Course

The course that participants were observed take 14 weeks excluding the first meeting and project development weeks In total course was 14 weeks long. The course took place in a computer laboratory at the Computer Education and Instructional Technology building, METU. The general structure of the computer lab can be seen in the picture below (figure 3.6). The computer lab was a U-shaped classroom with 12 desktop computers.

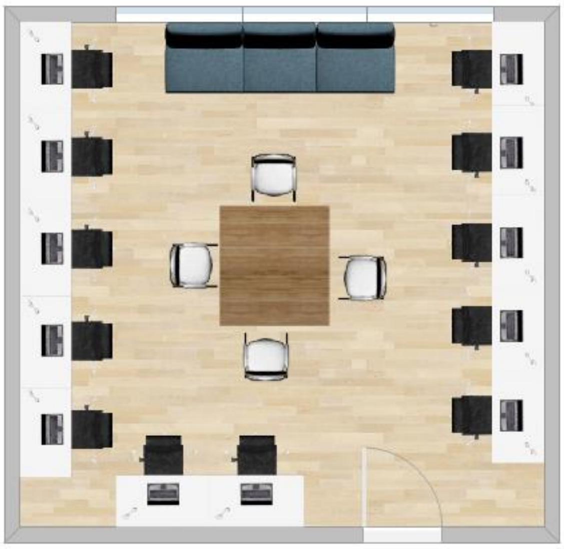


Figure 3.6 General Sitting Plan of the Classroom

As it can be seen from the figure 3.7, shape and structure of the classroom allowed students to see each other’s screens, to communicate, and to have discussion. Those features are not a must, however, it could be helpful.

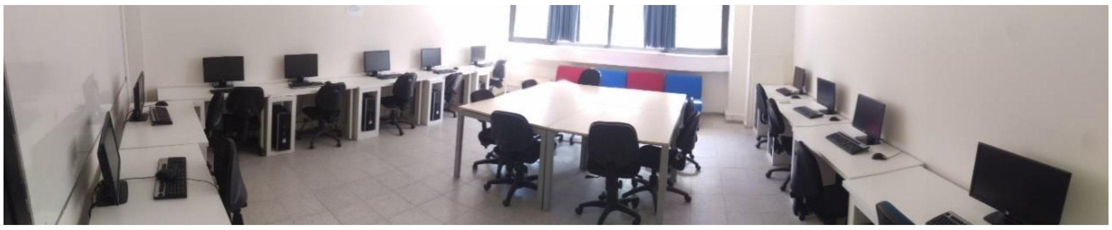


Figure 3.7 Picture of the Classroom

3.5 Participants of the Study and Sampling

Participants of this study were novice/non-programmer university students with basic computer skills. A course was opened up as an elective course in the Department of Computer Education and Instructional Technology (CEIT), Middle East Technical University (METU), so it was expected from participants to take the course willingly. For the first semester, 11 participants have participated in the course. For the second semester, 7 participants have participated in the course. In total, data collected from 18 university students. Large participants and assessing them could provide an inaccurate sense of students’ computing competencies. Therefore a small number of participants were selected to be worked with (Brennan & Resnick, 2012).

Fraenkel, Wallen, and Hyun (2012) stated that one of the most important steps for the research process is sampling process which involves the procedure and criteria to select the participants. “Sample selection in qualitative research is usually (but not always) nonrandom, purposeful, and small, as opposed to larger, more random sampling in quantitative research.” (Merriam, 2009, p. 16). Purposive sampling method used to select the participants as it was suggested for a qualitative research study. According to Patton (2002) strength of the purposeful sampling comes from in-depth understanding of a phenomenon through the information-rich cases which hold the important central knowledge serves to the purpose of the inquiry. Purposeful sampling is mostly preferred on the assumption that the researcher “wants to discover, understand, and gain insight and therefore must select a sample from which the most can be learned” (Merriam, 2009, p. 77). Fraenkel, Wallen, and Hyun (2012) suggested that for the generalizability of the studies, purposive sampling methods are not very decent to use. On the other hand, purposive sampling is appropriate to use in studies which solely rely on qualitative data (Fraenkel et al., 2012). Under the purposeful sampling method, one or more strategy could be applied to select the participants (Patton, 2002), In this study, a two-step strategy was applied in which researcher started with criterion sampling strategy and continued with the heterogeneous sampling strategy which will be mentioned later. According to Patton (2002), criterion sampling strategy under the purposeful sampling used for making sure of participants qualifies for the needed information. Therefore some criteria were applied to select the participants. Since the study aims to design an effective course for novice/non-programmers, selecting the participants from students with advanced programming skills could bias the results of the study. Additionally, participants should have basic computer skills to use the environment without getting help. To summarize there were be 2 basic criteria to select participants were:

1. Participants should have basic computer skills
2. Participants should be a novice or non-programmer

Registered students were selected based on their purpose and background. In addition to the criteria above, the common grounds of participants were as follows; (1) Students with the purpose of completing needed credit hours were eliminated to gather a meaningful data. (2) None of the students had previous experience regarding visual programming. It should be noted that the students from departments such as Electrical and electronics engineering or computer science were rejected, since they are not coherent for this study as participants. In addition, it was aimed to gather data from the students with different perspectives and from departments to reach a heterogeneous data. On account of the aim of reaching such data, heterogeneous sampling strategy was the second step. According to Patton (2002) heterogeneous (or maximum variation) sampling strategy is useful to reveal different viewpoints for each case and to reveal the shared pattern of participants from different cases. It was aimed to have at least one student who was good at programming for each semester. In addition, to ensure heterogeneity, students who have failed traditional programming and students with no programming experience were also selected. While students with programming experience could give an insight about the comparison of two different programming environments and instructional approaches and strategies that both courses used, students with no programming experience could help researcher to see the experience of first-time learners and the difficulties they have faced with when they are learning programming. Since this study included interviews with audio recordings, observations with video recordings, Facebook group discussions, the privacy of the participants will be taken care of by the researcher. Identities and personal data of the participants will be confidential throughout and after the study and protected by using code names. Participants, their departments and programming experience presented with their code names used in the findings part in table 3.2 below to get a clear understanding of the source of data in this study. The number after the letter "T" represents the term/semester they had participated, while the number after "S" used to create a unique code name for each participant.

Table 3.2 *Code Names and Characteristics of the Participants*

Code Name	**Department**	**Programming Experience**	**Gender**
(T1_S1)	Elementary Math. Education (EME)	None	Male
(T1_S2)	EME	None	Male
(T1_S3)	EME	None	
(T1_S4)	Computer Edu. and Ins. Technology (CEIT)	Good	Female
(T1_S5)	CEIT	Failed (C++)	Male
(T1_S6)	Elementary Science Education (ESE)	None	Female
(T1_S7)	ESE	None	Female
(T1_S8)	CEIT	Failed (C++)	Male
(T1_S9)	EME	None	Male
(T1_S10)	Physics	Introductory (C++)	Female
(T1_S11)	EME	None	Female
(T2_S1)	CEIT	Good	Male
(T2_S2)	CEIT	Good	Male
(T2_S3)	CEIT	Introductory (C++)	Female

Table 3.2 cont'd

(T2_S4)	Business Administration	Introductory (VB)	Male
(T2_S5)	CEIT	Introductory (C++)	Female
(T2_S6)	CEIT	Good	Male
(T2_S7)	EME	None	Female

Ratio figure of participants regarding their programming knowledge levels of before taking the visual programming course can be seen below. Majority of the participants have no background in programming or have taken and failed a programming course (56%) before. Additionally, the same number of participants who have taken and passed a programming course before this course.

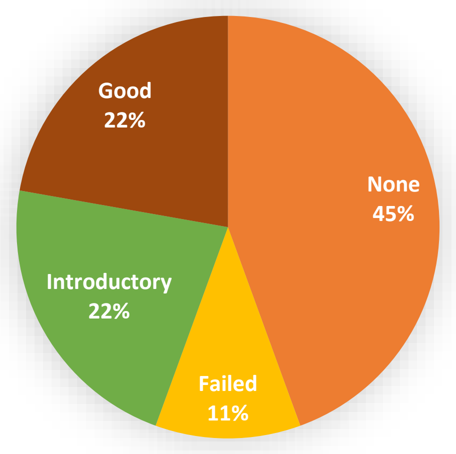


Figure 3.8 Programming Experiences of Participants before the Course

First analysis and design steps of a DBR study are crucial for the study (Richey & Klein, 2007). If any problem would have occurred in the first step, the researcher aimed to reach participants to replace dropouts as soon as possible. However, only one student dropped the course in the first semester due to the personal problems and took the course in the next semester. That student was not included in the data for both semesters as it was not included in the table below.

3.6 Researcher's Role and Potential Ethical Issues

Different than quantitative studies, the researcher is not separate from the research in terms of data collection, and interaction with the data source. According to Denzin and Lincoln (1998), the interaction of researcher with the researched phenomena and the data source shape the outcome by the perception of researcher which makes the researcher an instrument of the study. According to Merriam (2009) in real life, researchers are rarely total participants or total observers. In this study, the researcher was an all-time active participant of the group, however, neither a total participant nor a total observer. Creswell (2009) suggested that it is important to identify “the biases, values, and personal background, such as gender, history, culture, socioeconomic status” (p.177).

In qualitative research, the researcher should report any personal and professional information that may have affected the data collection, analysis, and interpretation of the study (Patton, 2002). Knowing the background and the position of the researcher is essential to understand the study and the outcome of the analysis. Researcher received his Bachelor’s Degree in Computer Education and Instructional Technology program. The researcher is also a Ph. D candidate and Research/Teaching assistant in Department of Computer Education and Instructional Technology, at a public university in Turkey. As of writing this dissertation, the researcher has 8 years of teaching experience in computer and programming education. Additional information about the researcher can be found at the Curriculum Vitae of the Researcher at the end of this thesis. Researcher was also the instructor of the course he gathered data from. In this way, he had the chance of observing participants at every critical point. Additionally, he had the freedom of improving and changing the course based on his observations anytime he needed. Moreover, after the classroom, students communicated with the instructor via the social media group. Therefore communication and observation of the students continued after the course hours.

The ethical side of a research study is also important for both research study and the role of the researcher. Before starting the study, researcher took the approval from Human Subjects Ethics Committee of the Middle East Technical University (Appendix-B). If the participants of the study are trusting and believing the researcher and the study, they will be more honest and open to communicate (Postholm & Madsen, 2006). Participants of the course/study were informed verbally and with a written informed consent form (Appendix-C) about the research study involving the purpose, data collection types, and process before the course started. Informed consent form is a form that invites, informs and explains participants what they are being taken part in before research starts (Postholm & Madsen, 2006). Additionally, researcher also notified students before video or audio recordings began. According to Creswell (2012), researcher should not create artificial data and share the outcomes with the participants. Researcher used quotes with the context and explanations to ensure the reader that data were valid. Additionally, participants were free to reach any outcome and publication of the research by reaching the researcher. Researcher shared all of the communication information with them and did not close the social media group to make certain that participants could reach the outcomes and researcher after the study any time they need.

Being the researcher and the instructor at the same time could also affect the criticism and the negative comments of the students. Researcher constantly encouraged students to put forward their positive and negative ideas during the course and in the interview sessions. Researcher also stated that any criticism about the course would be even more helpful to him.

3.7 Data Collection

In this part, data collection procedures and the study was examined. For the qualitative data collection, three different procedures were followed: Interview, observation, and documents. The data collection techniques used in a research study are determined by problem and purpose of the study, the researcher's theoretical orientation and selected sample (Merriam, 2009). The three data collection techniques followed to collect and analyze the data are the main techniques used in qualitative research studies (Fraenkel et al., 2012). Those procedures were examined in this part of this dissertation.

3.7.1 Data Collection Procedures

A distinct characteristic of qualitative research is that humans are both data source and primary data collection instrument. Since the qualitative research focuses on meaning in context, it mainly uses human to human interaction through interviewing, observation to gather and interpret the data (Creswell, 2007; Merriam, 2009).

Three basic way of reaching to the data in qualitative research are interviews, observations, and documents (Merriam, 2009). All of those three different types of qualitative data were collected to reveal the critical sides and to take the views of the students on the course and computer programming. There were 3 different data sources to answer the research question: Interviews, observation, and documents. Each data type could be used like quotes, screenshots, excerpts, and combinations of them to support the findings of the study (Merriam, 2009). Those three different types will be investigated one by one in the following parts of this dissertation. Central data source was a semi-structured interview conducted with students at the end of each semester. As Creswell (2007) and Mason (2002) suggested in qualitative studies, qualitative data mostly play a main role as the data source. The observations throughout the course and the documents which are mainly Facebook group discussions and messages of students supported the main data of this study.

3.7.1.1 Interviews

"An interview is a process in which a researcher and participant engage in a conversation focused on questions related to a research study. These questions usually ask participants for their thoughts, opinions, perspectives, or descriptions of specific experiences." (DeMarries, 2004, p. 54). According to Merriam (2009), the most common form of data collection in qualitative research, especially in the education field (in some cases the only one) is interview. Similarly, the main data source for this study was semi-structured interviews. A one-on-one interview was conducted with each student at the end of each semester.

The purpose of interviewing is to enter into another person's mind and see and understand their perspective (Patton, 2002). Mason (2002) explained that interviewing is the only way to access the mind of a person to gather interpretations and understandings. Even though the interviewer observed the participants throughout the course, it is essential to gather their opinions verbally through critical questions by interviewing. As Patton (2002) remarked interviews are also useful to reach data researcher cannot observe such as feelings, thoughts, and intentions. Moreover, interviewing is also an important tool to check the accuracy of data which was gathered through the observation (Fraenkel et al., 2012). According to Patton (2002), the quality of interviews heavily depends on the interviewer. Therefore, researcher/instructor himself interviewed with the participants. Since he did know all of the experiences, perspectives of the participants throughout the course, he easily managed the semi-structured questions. Bogdan and Biklen (2007) stated that good interviews are the ones in which participants talk freely and at ease about their opinions and answers.

Interviews were conducted after the evaluation of the students ended, and students were notified about that to let them know whatever their opinion was towards the course or the instructor, it will not affect their final grade. Moreover, participants were relaxed because they already know the instructor from the course.

A semi-structured interview was conducted to reveal the opinions and experiences of the students. According to Fraenkel, Wallen, and Hyun (2012), there are four types of interviews: Structured, semi-structured, informal and retrospective. Semi-structured interview was selected to provide a more flexible interview based on the different answers of the students. There are six types of interview questions: demographic, knowledge, experience, opinion, feelings, and sensory (Fraenkel et al., 2012). Interviews in this study included demographic questions, experience questions, and opinion questions since the research was foucsed on extracting the opinions and experiences of the students towards programming, course and the environment.

The most common form of recording the interview is tape-recording the data to ensure that data is preserved for analysis(Merriam, 2009). Interviews should be recorded audibly by a tape/voice recorder regardless of interview type (Creswell, 2007; Fraenkel et al., 2012; Patton, 2002). Therefore Interviews was recorded in audio format with two different voice recorder to make sure every word was recorded clearly. In addition to recording, interviewer took notes throughout the interviews. Patton (2002) suggested taking notes during the interviews for more than one reason such as helping the researcher to formulate new questions during the interviews, keeping the interview in the right direction, keeping the notes as a backup data etc. Interview questions can be found in the Appendix A. As it was stated before it was a semi-structured interview. Additional questions which are called probe and follow-up questions (Patton, 2002) were asked to students, according to the flow of the interview and the answers of the students.

3.7.1.2 Observations

Creswell (2012) defined the observation in qualitative research as "…the process of gathering open-ended, firsthand information by observing people and places at a research site." (p. 213). Observation technique was used in this study in order to examine the in-class activities, experiences, reactions of the students at the time they occurred. Observation is one of the useful research tools when it focused on solving a research problem and it provides an internal control mechanism to the checks and balances in producing trustworthy results (Merriam, 2009). It was also used to support other data sources to triangulate the findings of the study. Merriam (2009) also emphasized the observation's duty for triangulation to provide evidence, in coordination with interviewing and document analysis to strengthen the findings.

Advantages of using observation data in qualitative research highlighted by many authors. Merriam (2009) stated one advantage of observation over the other data collection types as it gives the researcher the chance of recording the behavior as it is happening. Observations have also an important role for interviews. Observations made before interviews could help to shape the interview questions (Patton, 2002). Without relevant data of the participants and the phenomenon, interviews and the follow-up questions used in the interview could be inadequate. Patton (2002) also lists the advantages of observation in the natural setting as [1] understanding the context, [2] making researcher more open and inductive, [3] providing researcher an opportunity of outsider view which he is aware of the routine the others could not, [4] learning things that participants would avoid talking in an interview. On the other hand, Mason (2002) pointed out another rationale for the observation data: the ontological and epistemological position of the researcher is essential for deciding whether to use observation or not. The epistemological standpoint of this study also suggests that gathering observational data is relevant to the study.

Role of the observer varies according to the classification in the qualitative research. Main two different observation types are participant observation and nonparticipant observation which are based on whether participants take the role as the full participant or observe the phenomenon without participating (Creswell, 2012; Fraenkel et al., 2012; Merriam, 2009; Patton, 2002). However, in this study more specific variation of observation type were used: Naturalistic observation which is more close to participant observation. Naturalistic observation consists of observing the individuals in their natural setting in such places as classrooms, athletic events, playgrounds etc. (Fraenkel et al., 2012).

Observation data mostly consist of unstructured text data and pictures taken during observations by the researcher (Creswell, 2012). In this study observations made by researcher/instructor himself, by taking notes about data-rich reactions, answer, questions and behaviors. In addition to notes for three weeks of the first-semester instructor recorded the video of the class to examine if his observations are accurate and if there was anything that he missed in general. After three weeks camera was removed from the classroom to remove its negative effect on the natural behaviors of participants. The researcher is the primary instrument for data collection in qualitative research as it was stated before which also could cause some problems regarding healthy observation data. Subjectivity, interaction, and inter-dependency between participants and observer are expected and could cause changes in behaviors of observed and observer (Merriam, 2009). This impact of the observer's presence on behaviors of the participants is known as observer effect (Fraenkel et al., 2012). Another problem to consider while gathering observation data is observer bias. Observer bias is the factor that interfere with objective observations such as emotions, prejudices, attitudes, personal interests, values of observers, hasty decisions etc. (Polit & Beck, 2003). Even though there is no strategy to completely remove those two negative effects, it is possible to minimize them. Fraenkel, Wallen, and Hyun (2012) suggested that spending a considerable amount of time at the data collection site could help to reduce both observer effect because participants would get used to observing and observer and observer bias, because getting to know the environment, and observees would strip the observers from their prejudices and made decisions. In this study, researcher spent time with participants every week about 3 hours in classroom, and more on Facebook group page to get to know them deeply to minimize the negative effects of observation.

3.7.1.3 Documents

Documents are also another form of qualitative data collection type. 'Documents' is the third major source of data and an umbrella term which refers to visual, written, audial, printed and other materials to be studied as the data source in qualitative research (Merriam, 2009). Documents ranged from official reports to personal diaries, e-mails etc. (Creswell, 2009). As the third type of data, documents were used in this study. As Patton (2002) remarked that documents has the potential of holding the rich source of information. Creswell (2012) also emphasized that documents are a valuable data source to help the researcher to understand the central phenomenon. In this study, Facebook group discussions, posts, comments, and messages were used to support the findings. Facebook group posts, discussion, and communication were encouraged by the researcher to let participants solve problems they encountered with the help of their peers. In addition to the written data, final projects/presentations of the students were also examined to reveal their knowledge and misconceptions, whether they were relevant with their statements in interviews or the observations of the researcher during the course hours.

According to Merriam (2009), documents are underused as a qualitative data source in qualitative research studies. Mason (2002) remarked that documents and visual data could present researcher an alternative angle on or add another dimension to the study, therefore used more commonly in social science research. Merriam (2009) also emphasized that having data sources like online data widens the scope of data available to the researcher, and provide it could establish unexpected communication forms and relationships.

The Facebook group comments and posts were also observed in real time, however, to see the bigger picture and to look at the data in a more holistic way logs were saved as pdf files. The presentations and the project files of the students were also examined to gather richer data. Documents primarily served the purpose of supporting the findings of interviews and observations.

3.7.2 Data Collection Process

As it was stated before, Design-Based Research consists of iterative cycles which are called micro, meso and macro cycles (Dai, 2012; McKenney & Reeves, 2012; Richey & Klein, 2007). This study also consists of Micro, Meso and Macro cycles. As it was mentioned before data collected from two different classes for two following semesters which formed the macro cycle of the study. The course was 14 weeks long including the first-meeting and project discussions, development and presentation weeks of the course. For the tentative outline and the topics of the course see table 3.3 below. Of course the products were the ones that shape the course and rather than focusing on the topic, the knowledge was built on top of the previous week’s.

Table 3.3 *Tentative Course Outline*

Date	**Topic**
1st week	First Meeting & Introduction
2nd week	Getting to know the environment
3rd week	Using Sprites and Creating Animations
4th week	Using Variables
5th week	If…Else Structure and Clock Use
6th week	Functions
7th week	Loops
8th week	Using Multiple Screens
9th week	Database and Lists
10th week	Working with Sensors
11th – 13th week	Project discussions and development
14th week	Product Presentations

Macro cycle consisted of 9 weeks. In each macro cycle, there are 2 meso and 4 micro cycles. Every step (or micro cycle) consists of analysis and exploration, design and construction, or evaluation and construction (see a planned macro process in Figure 3.9 below, adapted from McKenney & Reeves, 2012, p. 78). As McKenney and Reeves (2012) stated while researcher can collect data both in analysis and evaluation phase, and construct according to data he gathered in the design phase. It is not appropriate to collect data because in this phase researcher use data to reshape the product or model. Throughout the coherent cycles of the study, data were collected regularly, to be more specific each week as observation data. The researcher was also the lecturer of the course hence a constant observer of the course. Through 9 weeks researcher observed the participants each week, however main data collection was occurred at the end of the semester by conducting interviews with students.

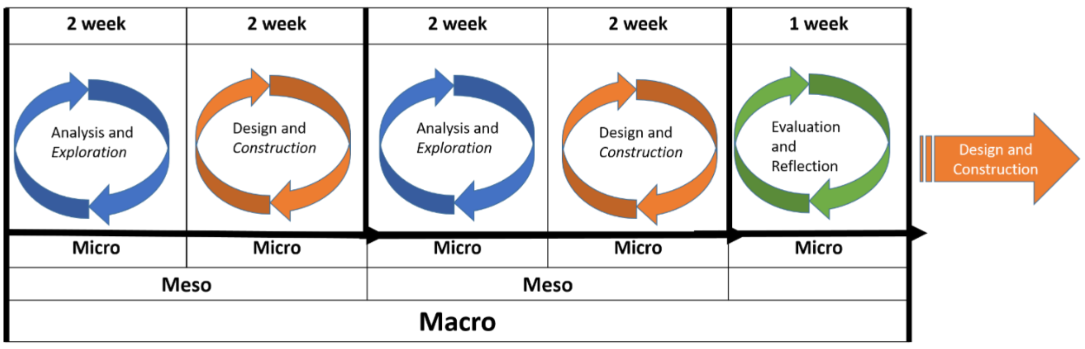


Figure 3.9 Process of DBR (adapted from McKenney & Reeves, 2012, p. 78)

Throughout the data collection process, researcher will observe the students and their behaviors. As it was stated before, naturalistic observation took part for observation. Researcher is also a part of the setting naturally and actually participates in the classroom that he observed and participants know that observations are being made (Fraenkel et al., 2012). During the data collection, in addition to notes and memory of observer, video recordings were also taken for the first three weeks since for that process researcher was still foreign to participants and their characteristics. According to Fraenkel, Wallen and Hyun (Fraenkel et al., 2012) recorded observation permit researcher to investigate the critical points repeatedly.

After the course ended interviews were conducted as it was stated before. Before the next course started researcher examined the gathered data which includes interviews, observation notes, observation videos, and Facebook group discussions to rebuild the course. The same process started over with new course started. Through the cycle and iterations, instructional strategies and suggestions were extracted.

3.8 Data Analysis

Data analysis in qualitative research studies is the process of searching and arranging the collected data such as interviews, fieldnotes, and other materials to understand and present them to others (Bogdan & Biklen, 2007). In this study, 3 different qualitative data source were used under the framework of Design-Based Research: Interviews, observations, and documents. Interviews with the students were the main source of data of this study. Therefore the main structure of the findings consisted of interview data. However, all of the qualitative data sources were analyzed similarly. According to Creswell (2009), analysis of qualitative data is like peeling off the layers of onion one by one to reach the core. Researcher of this study also used qualitative findings to reach the cores of the instructional strategies for a visual programming course.

At the end of each semester, interviews were conducted with every student as it was mentioned in the data collection part. An example of interview questions can be found at Appendix-A. Interviews were recorded digitally by two different recording device and transcribed verbatim. "Transcription is the process of converting audiotapes recordings or fieldnotes into text data." (Creswell, 2012, p. 239). Interviews were transcribed by the researcher to get a general picture of the data, and insights from the data as Merriam (2009) suggested, rather than hiring somebody to do it. Even though those transcriptions are only raw data of the study, it shed a light of where the study is heading. A general mistake that is being made by the researchers is waiting for data collection to be completed, and then start to analyze. Merriam (2009) stated that this mistake leads researchers to drown in the hundred pages of data. She suggests that the researcher should start analyzing data right after the first data were collected. As Patton (2002) declared that there is no strict line between data collection and analysis in qualitative research. After transcribing the data, all of the transcriptions were printed and read by the researcher. Patton (2002) suggested that reading and checking the transcriptions is beneficial to get a sense of the whole. Data were analyzed with NVivo 10 software which is a qualitative data analysis tool. Nvivo offers a complete toolkit for both visual and textual qualitative data (Creswell, 2012). Qualitative data analysis tools help researchers to read, organize, review, and categorize the qualitative data acquired from texts, pictures, or videos more easily and faster. (Creswell, 2012; Patton, 2002).

According to Creswell (2012), qualitative data should be reduced and transformed to make it more accessible, understandable and be able to draw out conclusions. Meaning of reducing and transforming data is focusing on the important parts of the answers and transform them into grounded themes and theoretical conclusions (Creswell, 2007). Qualitative analysis usually starts as an inductive process which is a primary characteristic of qualitative research that researcher should figure out the possible patterns from the data (Patton, 2002). The most common technique to analyze the data among researchers is coding (Fraenkel et al., 2012) in which the researcher of this study used the same. In this study, interviews were analyzed and transformed into codes to create a frame and make sense of the data instead of dealing with raw hundreds of pages of data. Coding is a process in which researcher make sense of data and label them the segments with short names representing the information in that segment (Creswell, 2012). The form of coding that inductive and open to any type of information is called ‘open coding’ generally (Merriam, 2009). According to Lincoln and Guba (1985) a code should have two characteristics: First, it should be to the point of the study that it should provide some understanding of the research study; second, the code should represent the data by itself without a need of additional information other than the broad context of the study. Codes were checked for consistency and accuracy to see their similarities and differences before creating the themes. This technique is called constant comparison which is a method used in the creation of the codes to check within and between cases (Gibbs, 2007). After creating the codes, codes were formed into categories to generate the themes of the research study. Themes are the categories which were constructed after gathering similar types of codes under them (Merriam, 2009). Themes are a core element of qualitative research to form a major idea by collecting codes together (Creswell, 2012). The process of the data analysis can be seen from the figure 3.10 (adapted from Creswell, 2009). This process also coherent with the epistemology of Desing Based Research. DBR also suggests that knowledge should be constructed in the beginning and re-constructed and validated in the light of new data.

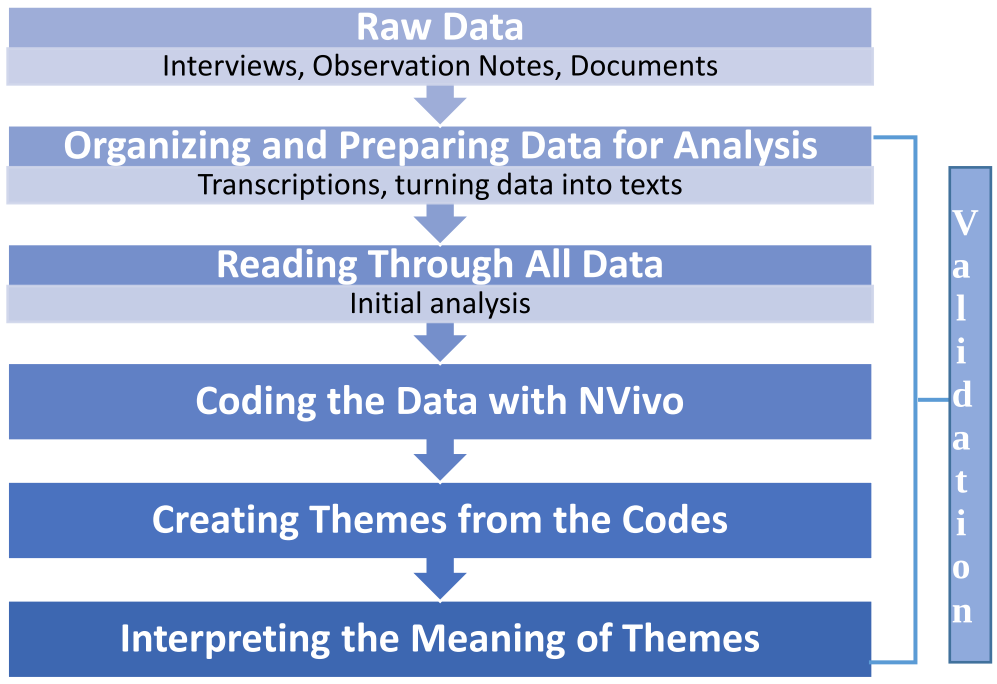

Raw Data
Interviews, Observation Notes, Documents
Organizing and Preparing Data for Analysis
Transcriptions, turning data into texts
Reading Through All Data
Initial analysis
Coding the Data with NVivo
Creating Themes from the Codes
Interpreting the Meaning of Themes
Validation

Figure 3.10 Data Analysis Process (adapted from Creswell, 2009)

Observation notes that were taken during the class and the video recording of the course were also analyzed using the same process and tools with the interviews. Another source is documents which were consisted of Facebook group posts and comments. Patton (2002) suggested that deductive approach can be used in the confirmatory stage of analysis. Coherently, in the final/confirmatory part of the coding process, remaining codes that were not placed under any categories, deductively tested if they fit into one or could be generated into a new category. After creating the themes, the themes were supported with the results of other data sources such as observation notes and Facebook group posts and comments. Notes from observation during the class, transcriptions from the critical points of video recordings and screenshots of every post and comment were also transcribed and analyzed to be used for confirmatory part of the analysis. Analysis of those three different data sources was also used to make sure that the findings are accurate. Triangulation method was used which is a process of corroborating evidence by using data from different individuals, different sources or methods of data sources (Creswell, 2012), which will be investigated further in the following part.

3.9 Trustworthiness of the Study

In this part, the validity and reliability of the study will be investigated. According to Bogdan and Biklen (2007), there is no standard for testing the validity and reliability in qualitative research studies. Additionally, in qualitative studies, validity and reliability, are named and secured differently. Merriam (2009) stated that "credibility, transferability (generalizability), dependability substitutes for internal validity, external validity, reliability." (p. 211). Both terms will be used under this section. In true qualitative fashion, as well as there is no sole truth or objective reality regarding collected data, analysis and analyzer of the data would also have different aspects. Although the paradigms and methodologies enlight the path, each person makes sense of the underlying philosophical influences in his or her own way (Merriam, 2009). This could be seen as researcher bias or subjectivity that threats the validity. Merriam (2009) explains that identifying and monitoring the subjectivity and their effect on the study is an applicable and suggested strategy rather than trying to eliminate these biases or 'subjectivities'.

3.9.1 Triangulation

Triangulation is one of the main and most well-known strategies to ensure the trustworthiness (specifically credibility/internal validity) of a qualitative study (Merriam, 2009). According to Creswell (2012), triangulation is the process of corroborating evidence from different individuals, types of data, or methods of data collection. Triangulating the outcome by using different data sources and methods would build a coherent justification for the themes created (Creswell, 2009). Patton (2002) emphasized that the purpose of the triangulation is not reaching to the same result from different sources, the point of using triangulation is to test the data for consistency. As it was mentioned before, three different types of data were collected in this study: Interviews, observations, and documents. Fraenkel, Wallen, and Hyun (2012) and McKenney and Reeves (2012) also suggest using various instruments and methods to collect data from the real-life setting enhance ecological validity of the study by triangulating the data. Patton (2002, p. 556) on the other hand, reported that there are four kinds triangulation to contribute to verification: (1) Methods triangulation, (2) Triangulation of sources, (3) Analyst triangulation, (4) Theory/perspective triangulation. While not all of the triangulation kinds applicable to all types of studies, using diverse triangulation methods could be helpful to ensure the validity of the study. In this study, as well as collecting from different sources, data were collected from two different semesters and from students from different departments due to the nature of design based research framework.

3.9.2 Intercoder agreement

The intercoder agreement is a strategy to ensure the reliability of the study based on the use of multiple coders to analyze transcript data (Creswell, 2007). Generating codes from the raw data is one of the most crucial points of the study. However, looking from a single point of view could lead to unreliable outcomes. Checking or generating the codes by different researchers is essential for the reliability of the research study (Miles & Huberman, 1994). Intercoder agreement did not mean researchers used the same passage for the exact same code, it rather means that researcher understands the similar meaning from a passage of text and codes it similarly (Creswell, 2009).

Firstly three different interview transcriptions were handed over to two researchers. Researchers were also Ph. D. Candidates and from Computer Education and Instructional Technology, Middle East Technical University. The purpose of the study, the characteristics of the participants, and methodology of the study were explained to the researchers. Coders were given 1 week to analyze the data. After the analysis completed, coders and the researcher set up a meeting to cross-check their codes. At the start of the meeting, coders checked and compared their codes with the researcher. According to Miles and Huberman (1994), agreed code to total code ratio should be equal to or greater than 80% even if it is ok to reach 70% at the first meeting. After the calculation, it was revealed that the similarity ratio with one researcher was 81% and it was 83% with the other one. At the second round, coders discussed the differences and disagreeing codes to reach an agreement through a codebook. Some of the code names were changed and one code was added. Additionally, one theme name was changed (from ‘recommendations for the course’ to ‘dynamics and evaluation of the course’) and the old theme name was converted into a sub-category (Computational Thinking).

3.9.3 Rich/Thick description

Rich (or thick) description is a validity strategy to transport readers to the setting and to make results more realistic and richer by giving them as detailed as possible information about the setting (Creswell, 2009). Creswell (2007) also stated that this strategy allows readers to make decisions regarding transferability. In this study, researcher also aimed to explain the setting, themes, context, process, and the findings of the study. Merriam (2009) does not only suggest the rich description as a strategy to enhance the transferability of the result to other settings but also explains it as one of the main characteristics of qualitative research.

3.9.4 Real-life Setting / Prolonged Time

To ensure generalizability (external validity in a classical term), researcher should collect data from natural work settings and representative samples (Kelly, 2004; Richey & Klein, 2007). McKenney and Reeves (2012) state that transferability of a design based research study is increased when conducted under real-world conditions. Additionally, Creswell (2009) reported that the more time researcher get to spend in the actual setting, the more chance the more accurate or valid the findings will be, because of the in-depth understanding of the setting and participants. In this study, as it was mentioned before, researcher was also the instructor of the study and he observed and participated the classroom setting at least 3 hours a week, and much more in the social media group.

3.9.5 Peer Examination

Peer examination is a process in which knowledgeable peers about the topic and the methodology examines the manuscript and gives recommendations; as Merriam (2009) put forward that all of the dissertations have this strategy, thanks to the Ph. D. advisor and thesis monitoring committee. Additionally, researcher shared the study with his colleagues occasionally to make sure that the steps he followed were decent. This strategy by involving other people beyond the researcher's perception adds validity to the study (Creswell, 2009).

3.9.6 Researcher's role

Providing researcher's role (position) is essential in a qualitative study to understand the analysis, description, and the context. As the researcher gives detailed information about himself, regarding his experience, background, culture, etc., the bias of the researcher would be clarified and provide a more honest and open result (Creswell, 2009). Researcher's role was provided regarding his background and experience. By providing such information, readers could see the researcher's position, interpretation and approach to the study and clarify the bias of him/her (Creswell, 2007).

3.9.7 Negative Information

Presenting the negative information that was counter to the themes improves the credibility of the study, since the real world composed of opposing information (Creswell, 2009). In the analysis phase, researcher looked for negative versions of the code for each turn as Miles and Huberman (1994) suggested. In this study, opposing codes were even presented, since some of the students disagreed on some aspects of the course. Gibbs (2007) suggested that discussing negative cases fall outside of the pattern, providing contradictory descriptions could show the complexity of the data and identify the nature of the study. Providing both sides not only increase the credibility of the study but also let readers to see different perspectives regarding a specific topic. Even though the study were not to prove a hypothesis, giving a contradict point would help for further studies regarding the topic.

CHAPTER 4

RESULTS

The analysis of the interviews, observation notes, videos, discussions and posts of the Facebook group and the final project products of the students revealed five themes. The themes are Communication, Contributions of the course, Motivation, Programming and Programming Environment, and Dynamics & Evaluation of the Course. Those themes will be examined one by one with the support of interviews, observations, and documents. The coding table including themes and sub-themes can be found in Appendix D.

4.1 Communication

One of the themes emerged from the interviews was communication. Since this theme includes the communication of students, in addition to the interviews, Facebook logs and observation notes are also included. Students found communication essential both between students and student-instructor for the course. Two sub-themes appeared under the communication and interaction theme: Communication with the Instructor and between students, and Communication Medium. Especially, communication medium was seen very important for the students. Firstly, communication with the instructor sub-theme will be examined.

4.1.1 Communication with the instructor and between students

According to the interviews, 4 of 17 students stated that the communication with the instructor was very crucial. Both in and out of the course hours, students feel better if they know they can reach out to the instructor any time they need him/her. Student T2_S3 linked her not having any difficulty throughout the course to good communication with the instructor. She defined the proper communication with the instructor as one of the best sides of the course since she can ask a question any time she wanted.

> Communication with you (instructor) and us was good. When we have a question or a problem, you always helped to solve it. That was nice. Because communication with the teacher is very important to me... Since our communication with you was good, I did not have any difficulty. This was the side I liked most for me. Strong communication and being visual… (T2_S3)
>
> *Sizin bizle iletişiminiz iyiydi. Her sorumuz olduğunda problemimiz olduğunda yardım ettiniz çözdünüz. Bu güzeldi. Çünkü öğretmenle iletişim çok önemli benim için… Sizle iletişimimiz iyi olduğu için sorularımıza cevap bulduğunuz için sıkıntı yaşamadım. En sevdiğim yönü buydu. İletişimimizin kuvvetli olması, görsel olması… (T2_S3)*

Good and continuous communication with the instructor could affect the motivation of the students not only towards the course but also with the topic. According to the recorded videos and the observation notes of the researcher, even in the topics students have had difficulties with, students asked instructor and each other without hesitating. In addition to asking questions when needed, student T1_S2 also stated that informal and sincere communication with the instructor affected their attitude towards the course when asked what he liked about this course.

> Your communication with us was good. As I have said, in the course, it was not like instructor and student, more like a big brother, little brother. This semester was very nice for us. (T1_S2)
>
> *Sizin bizle iletişiminiz iyiydi. Hani dediğim gibi. Dediğiniz derste de öğretmen öğrenci değil de abi kardeş gibi oldu. Çok güzel oldu bence bu dönem bizim için. (T1_S2)*

Students also found communication with each other important. In both classes for two terms, students could talk with and help each other during the course hours. Moreover, after course hours they can discuss the topics through the Facebook group of the course. One of the students (T2_S3) defined their communication with each other as fun. He also mentioned that he examined what others did from the Facebook group.

We were talking to each other and such. It was fun because of this. I looked at what others did. I did not use them but to see how they did it how they solved it. (T2_S3)

Kendi aramızda şey yapıyoruz konuşuyoruz falan filan. Onun için eğlenceliydi. Diğerlerinin yaptıklarına baktım. Kullanmadım ama nasıl yapmışlar, acaba şunu nasıl çözmüşler? (T2_S3)

If students do not have good communication with the instructor, their attitude towards the course and the topic could be affected. When one of them has a question in mind, they want to be able to ask it anytime they need. They think that it is beneficial to close the gaps and to ask what they did not understand thoroughly. Students also see that as a positive feature of the course as students T2_S6 and T1_S3 explained. T2_S6 compared the course with another programming course he took regarding the communication.

(Regarding other programming courses' lab) Because in his lab, the professor comes and says "do this" and leave. When you ask a question, there is a slight chance to get an answer to your question. (T2_S6)

Çünkü labında hoca geliyor şunu yapın diyor gidiyor. Soru sorunca cevap alma ihtimalimiz çok düşük. (T2_S6)

I think it was very active. I don't think it has any shortcoming. We were constantly in communication. You always answered our questions… (T1_S3)

Bence çok aktifti ben herhangi bir eksiği olduğunu düşünmüyorum. Zaten sürekli iletişim halindeydik. Siz sürekli hani cevap veriyodunuz. (T1_S3)

It is important to note that positive attitude towards and communication with the instructor are other sources of motivation as expected. Instructor has to make sure that students would not be bored by the course or felt distant from the instructor. Communication should not be limited to course, but it should continue in social media as a continuous icebreaker between instructor and students. Interacting or even just posting on the social media could be a good option for the crowded classes to help shy students to be motivated towards the course. T1_S2 has also emphasized the communication, and the atmosphere of the course is affecting their attitude towards the course.

> Your relation with us and your personality, and the content of the course, etc. I mean course was interesting. We came to class willingly without getting bored… Entertaining… I mean not entertaining, you cannot expect entertainment from a course, but it is important not to be boring… We were laughing, we were having fun, and we were working hard (T1_S2)
>
> *Sizin bizle ilişkileriniz bi de kişiliğiniz dersin içeriği olsun şey olsun ilgi çekiciydi yani ders. Sıkılmadan seve seve geldik. Eğlenceli… Eğlenceli değil de hani dersten eğlence bekleyemezsiniz de sıkıcı olmaması önemli bence… Gülüyorduk, eğleniyorduk, uğraşıyorduk (T1_S2)*

Other than social media, feeling comfortable when asking a question to the instructor was also important for students to be motivated towards the course as student T1_S8 remarked.

> We felt comfortable. Asking you questions in person was comfortable. (T1_S8)
>
> *Çok rahattık. Birebir sizle çok rahat sorular sorduk. (T1_S8)*

Observations revealed that open communication with instructor directly related to motivation of the students towards course. One of the students emphasized that communication throughout the course and creating products made the course enjoyable and stated that “even if the course is very early in the morning, we did not think that as a course. This is a fun thing to do.” Observation notes also showed that most of the students felt free and started asking questions without hesitating from the second week of the course.

According to observation notes, students who knew each other started to help each other in the first week. Peer support was automatically started. However, some of the students who did not know each other was not communicating in the classroom. They finished the tutorials without communication. Peer support just needs a little encouragement for students who do not know each other. Using an informal communication medium and sincere communication during the class could break the ice and makes it easier for all students to know and help each other. Also, observation notes from the second week of the first term showed that one student who finished earlier than her friends showed her product to her friends before leaving the class and helped them willingly. Communication during the course occurred regularly which was driven by need oriented information sharing. A student wished for copy feature during the course and instructor showed him the similar ‘duplicate’ feature. As it did occurred for the duplicate feature, students kept sharing their information with each other instantaneously. Another example from observation notes shows that communicating during the course inspired benefitting from peer experience. One student deleted the entire function of blocks by pressing delete button. While looking for a solution, he explored the checkpoint feature which is similar to “Save as” feature of the traditional software. After the exploration, his friends benefitted from his knowledge and they also used checkpoint feature. Observation notes showed that help from peers was accepted more easily. Students even formed a group to overcome the problems of their friends.

Free and open communication inside and outside of the classroom should be encouraged so that students who have difficulty understanding the topic could get instant help from his peers. Encouraging peer-support was one of the important strategies came up from the observations. Encouraging peer-support was named as support-buddies strategy in which instructor will assign some students to each other to help the needed one. Support-buddies strategy will be examined further in the course dynamics and the discussion part.

While communication with the instructor is important for students, the communication medium was also as much important as the former. In addition, the communication medium also holds the potential to improve student-instructor and student-student communication. Constant communication with the instructor should be the standard for an introductory course. Next codes will include the influence of communication medium and opinions of students towards the use of the medium including communication with the instructor and between students.

4.1.2 Communication medium

One of the main sub-theme of communication and interaction theme is communication medium since the communication medium which was a Facebook group for this course was very important and influential to construct a stronger communication outside of the course hours. Observations throughout the course, interviews with the students and experience of the researcher as an instructor revealed that one of the essential things about the communication medium was being common among students. Communication medium was determined by the consensus of the students. At the beginning of the first class, instructor collected information about their communication media usage habits via a short survey to ensure that communication medium was used regularly out of the class hours. In the survey, a wide range of communication media was offered to students including email, LMS messaging, Facebook, Twitter, or any medium they want. All of the students agreed on Facebook as the communication medium of the course. Social media use is widespread among Turkish people including university students. One of the most popular one among others is Facebook with more than 48 million users (Kemp, 2017). All of the students participated the course stated that they already have Facebook accounts and 17 of 18 students indicated that they actively use Facebook in their spare times. The one student who had not actively used Facebook said that he opened the account for another course he took from the department. Use of Facebook in education is getting popular among researchers (Manasijević, Živković, Arsić, & Milošević, 2016). In this course, students were the deciders of the communication medium, and after the implementation, they were the ones who evaluated the use of Facebook in the course. Their opinions and views were taken towards the use of communication medium. Codes under this sub-theme are "positive sides of the communication medium" and "negative sides of the communication medium". Codes under those categories were examined further one by one.

4.1.2.1 Positive sides of the Communication Medium

Popular and Common Use among Students

Students had both positive and negative opinions about the use of the social media in the course. Majority of the students (14 out of 18) have stated their positive opinions towards using the Facebook. Firstly, students declared their positive views about how the Facebook group was suitable for the communication, in general. One of the positive views about Facebook was that being very popular and common so that everyone could reach out to each other and students liked using it. Student T1_S1 and T1_S6 explained` that it was advantageous to use a medium which is popular among students.

> The Facebook group was good in terms of communication… Almost everyone uses Facebook and use a lot. (T1_S1)
>
> *Facebook grubu bence iyiydi haberleşme açısından… Hemen hemen herkes Facebook kullanıyor, bir de bayağı kullanıyor. (T1_S1)*
>
> It was good. I mean more communication… was established. We were certainly sign-in to Facebook. (T1_S6)
>
> *İyi oluyordu. Yani daha çok iletişim… Daha çok sağlanıyordu. Mutlaka giriyorduk çünkü Facebook'a. (T1_S6)*

Student T2_S1 also emphasized the effectiveness of using the Facebook group as the communication medium since it is popular.

> It is effective. There are lots of people use Facebook nowadays. (T2_S1)

According to the observations throughout the course and during the examination of the Facebook group, it was clear that selecting a commonly used medium had a positive impact on more than one point. The first point was that since the students are familiar with the environment and experienced about how to use it, there was not any learning step for them. Second, Facebook is a warming environment, since the students were using Facebook in their spare time. Third, students instantly notified about the homework, announcements or the posts. Student T2_S7 also had a similar opinion about the positive aspect of the frequent use of the communication medium.

> It was much better in my opinion. We log in Facebook more so we can see them (posts, announcements) more. (T2_S7)
>
> *Bence çok daha iyi oldu. Öbür türlü face'i daha çok girdiğimiz için daha çok görebiliyoruz. (T2_S7)*

Some of the instructors use the university's course management system: METU Online. Students stated that they do not log in to METU Online regularly. It was only used for some of the courses students take. So, as student T1_S7 put forward that they would not prefer to use a system they do not even want or need to log in. If they do not log in, they will not hear the announcements, and they will not be mentally ready and active about the course.

> In my opinion, the Facebook group was good, it was nice. I mean, I don't sign-in to METU Online regularly, but Facebook is always open. I see it all the time. (T1_S7)
>
> *Bence Facebook grubu iyiydi, güzeldi. Yani sürekli, mesela Metu Online'a girmiyorum ama Facebook açık oluyor sürekli görüyorum. (T1_S7)*

Common use among students makes the other students reachable anytime. Being an active user of the communication medium out of the course hours was affecting the attention of the students towards posts and announcements from the course. Students were thinking that using Facebook as the communication medium is both practical and convenient for them since they use it all of the time. Using a communication medium that was used by students in their spare time could motivate them in terms of keeping them updated. For students who are not active regularly on Facebook, "Notification" was advantageous regarding being more noticeable. "Notification" feature of the Facebook has kept student updated about the course. Student T1_S1 was thinking that notification feature helped the active communication of the course, even when students were not using the Facebook group.

> Even when they are not using, a notification pops up on the phone when someone shared something on Facebook. (T1_S1)
>
> *Kullanmasa bile, telefonuna bildirim geliyo Facebook'tan bir şeyler yazılınca. (T1_S1)*
>
> An announcement can be spotted faster or attracts my attention more. (T2_S5)
>
> *Bir duyuru daha hızlı ulaşıyor açıkçası, ya da daha çok dikkatimi çekiyor. (T2_S5)*

Student T2_S5 and T1_S2 were also thinking that Facebook group was faster, immediate, and more noticeable.

> Yes, it happens directly and instantly, we can (learn) who did see, who did not see, what to do… (T1_S2)
>
> *Evet direkt anlık oluyor. Kimin gördüğünü, kimin görmediğini, ne yapman gerektiğini, hemen şey yapabiliyoruz. (T1_S2)*

While the students saw notification feature positively, overusing it could cause negative attitudes towards the page of the course. Posting too much could draw students away, bore them with unnecessary information, or tire them as student T2_S6 stated. If students feel bored because of too many postings, they could just turn off the notification, and that will eliminate the whole advantage of the notification feature.

> It was active, I mean it was nice. I mean it was active and not tiring. For example, I turn off the notifications in other courses. (T2_S6)
>
> *Çok etkindi yani güzeldi. Mesela şeydi, etkindi ve yorucu da değildi. Mesela ben çoğu ders şeyinde, diğer derslerde falan bildirimlerini kapatıyorum. (T2_S6)*

Multi-directional, Interactive, and Open Communication

Some students compared Facebook group to the other communication media they used before in other courses as it was mentioned in the previous code. According to students and observations of the researcher, the majority of the other instructors use email to communicate with students. Therefore, students mostly compared the Facebook Group with email, since email is a more conventional method for the students regarding other courses. When student T2_S5 compared the Facebook group with email, he emphasized the interactive and multi-directional features, while student T2_S7 found the Facebook group more practical.

> It is faster, and a lot more interactive than email in my opinion. Everybody can talk with each other or everyone can speak up their mind. (T2_S5)

> *Daha hızlı ve mailden çok daha interaktif geliyor bana. Herkes birbiriyle konuşabiliyor ya da herkes fikrini söyleyebiliyor. (T2_S5)*
>
> Otherwise, it took a lot of time to check my e-mails. Therefore, I think using Facebook is better. (T2_S7)
>
> *Öbür türlü, benim maillerime bakmam çok uzun zaman alıyor mesela. O yüzden, bence face kullanmak daha iyi. (T2_S7)*

Student T2_S5, in additon to his previous idea, was thinking that Facebook group should be used as the main communication medium in other courses too.

> It was good. I think that it should be used in plenty of courses. (T2_S5)
>
> *İyiydi bence. Birçok derste olması gereken bir şey. (T2_S5)*

Using a Facebook group provides a multi-directional and open communication in which students could, not only communicate with each other and instructor in private, but also open communication like a discussion page. Therefore, other students can see and learn from each other mistakes and solutions. Student T1_S8 compares using the Facebook group instead of email and states that it allow them to learn from each other's mistakes and questions.

> Using Facebook was seriously helpful and nice because we learned some things from our classmates' mistakes and questions. However, in other courses, our friends communicate directly via email and learn by himself. I mean we have a little chance of learning those. (T1_S8).
>
> *Facebook 'u kullanmak cidden çok kolaylık sağladı ve çok güzel oldu. Çünkü diğer sınıf arkadaşlarımızın hataları ve sorularında da biz bişeyler öğrendik orada. Ama diğer derslerde diyim arkadaş projede birebir irtibata geçip mail yoluyla sadece kendisi öğreniyodu yani biz öğrenme şansımız zor oluyodu. (T1_S8)*

Student T1_S1 found using Facebook group beneficial for not only communicating with each other but also seeing the posts of each other. Student T1_S5 also found the discussion-like communication helpful to learn from each other's questions. Open communication between students could be very helpful for a complex topic like programming.

> We can also talk with our friends and see their posts. (T1_S1)
>
> *Bi de arkadaşlarımızla da şey yapabiliyoduk işte konuşabiliyoduk paylaşımları görebiliyoduk. (T1_S1)*
>
> One of the things I liked about this course was our friends' asking questions to each other on Facebook. (T1_S5)
>
> *Dersin sevdiğim yönleri şey mesela güzeldi facebookta arkadaşların birbirine soru sorması. (T1_S5)*

Another comparison point for communication media was about the atmosphere of the environment. Some students prefer the informal structure of Facebook to e-mail. They found e-mail more formal and strict as student T1_S1 stated.

> It was good. I mean we could have some difficulties with email. E-mail is a more formal environment. (T1_S1)
>
> *Bence iyiydi. hani o olmasaydı zorlanabilirdik maille falan email biraz daha resmi ortam oluyor. (T1_S1)*

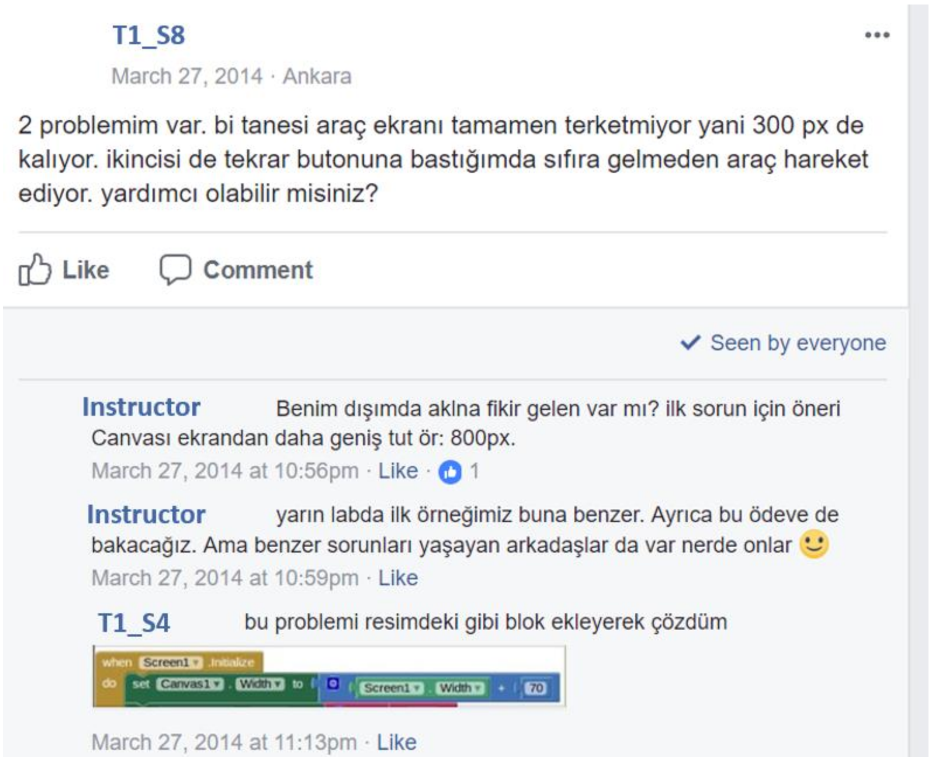
In addition, they want to have the chance of instant and constant communication with the instructor. The Facebook environment could help to break the ice between students and the instructor. Student T1_S10 and T1_S11 linked the communication with the instructor opportunity because of the informal environment of the Facebook group.

> So that it was the Facebook environment, we were constantly in communication with you. (T1_S10)
>
> *Şimdi Facebook ortamı olduğu için sürekli sizle iletişim halindeydik. (T1_S10)*
>
> I mean, in my opinion, the group was sufficient, all in all… I mean, I think there was a helpful environment in there. You were always active in there. (T1_11)
>
> *Yani bence grup gayet yeterliydi sonuçta... Hani yardım ortamı oldu hocam bence orada. Siz sürekli aktiftiniz orada. (T1_11)*

In addition to those positive views, some students (T1_S2 and T2_S3) stated that Facebook leads to better communication hence building stronger social interaction and friendship between students. Students found Facebook as a warmer environment. Informal communication between students through the social media leads to a warmer environment, which could affect the attitude towards the course.

> Most of us are on Facebook like 5-6 hours a day. We knew it already what's in there. There was a warmer environment in our Facebook Group. (T1_S2)
>
> *Facebook'a günde çoğumuz günde 5-6 saat giriyoruz. Biliyoruz zaten ne olduğunu orda. Bir de daha sıcak bir ortam vardı Facebookdaki grubumuzda. (T1_S2)*
>
> It led to both a social friendship with people in there (course's Facebook page) and establishing instant communication with you. (T2_S3)
>
> *Hem oradaki insanlarla bir sosyal arkadaşlık kurmamıza sebep oldu hem sizle anında iletişim kurabilmemize sebep oldu. (T2_S3)*

T1_S5 stated, keeping students communicating and asking questions to each other openly, was one of the favorite features for the students about the communication medium. As one of the examples can be seen from the figure 4.1 below, students asked questions to their friends through the wall of the Facebook group, when they had a problem with their homework. After the instructor's answer, another student who have had the same problem while doing homework, shared his solution with a screenshot of his way of solution.


T1_S8
March 27, 2014 · Ankara

2 problemim var. bi tanesi araç ekranı tamamen terketmiyor yani 300 px de kalıyor. ikincisi de tekrar butonuna bastığımda sıfıra gelmeden araç hareket ediyor. yardımcı olabilir misiniz?

Like Comment

Seen by everyone

Instructor Benim dışımda aklna fikir gelen var mı? ilk sorun için öneri Canvası ekrandan daha geniş tut ör: 800px.
March 27, 2014 at 10:56pm · Like · 1

Instructor yarın labda ilk örneğimiz buna benzer. Ayrıca bu ödeve de bakacağız. Ama benzer sorunları yaşayan arkadaşlar da var nerde onlar 🙂
March 27, 2014 at 10:59pm · Like

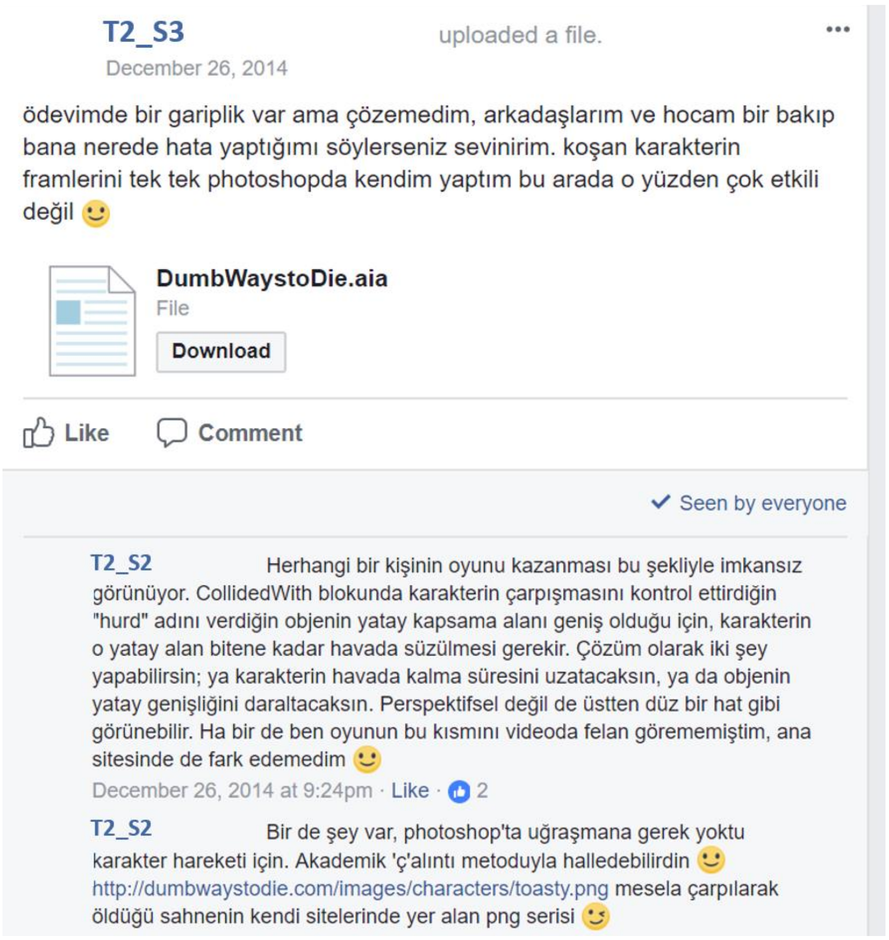
T1_S4 bu problemi resimdeki gibi blok ekleyerek çözdüm

```
when Screen1 .Initialize
do  set Canvas1 . Width to Screen1 . Width + 70
```

March 27, 2014 at 11:13pm · Like

Figure 4.1 Peer Help Using Facebook Group

Another example of students help through the communication medium can be seen below in figure 4.2. Student asked a question about her problem in homework and before instructor saw it, another student came up with the probable solutions to the problem. Through the notification whoever saw the post first he or she could help her or his friend. If students are motivated to help each other, both helper and the beneficiary would learn more effectively and permanently. While sometimes students helped each other, sometimes providing help for more complex problems could need some encouragement.


T2_S3 uploaded a file.
December 26, 2014

ödevimde bir gariplik var ama çözemedim, arkadaşlarım ve hocam bir bakıp bana nerede hata yaptığımı söylerseniz sevinirim. koşan karakterin framlerini tek tek photoshopda kendim yaptım bu arada o yüzden çok etkili değil 🙂

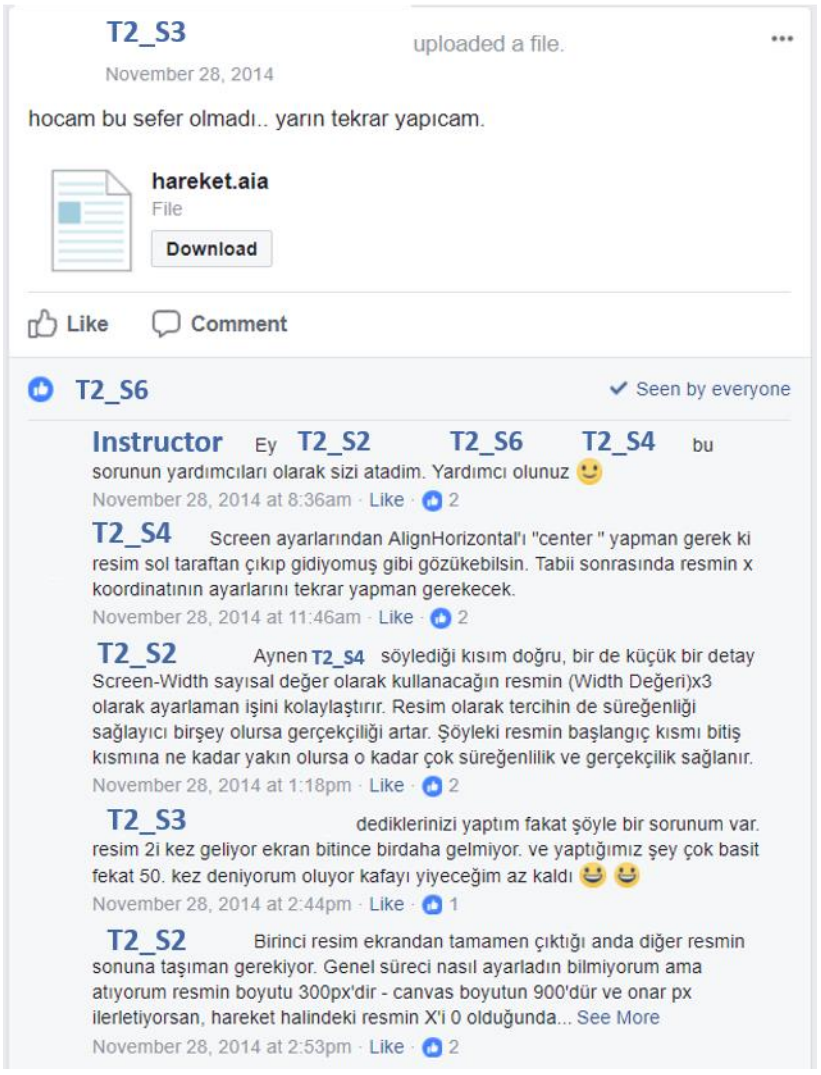
DumbWaystoDie.aia
File

T2_S2 Herhangi bir kişinin oyunu kazanması bu şekliyle imkansız görünüyor. CollidedWith blokunda karakterin çarpışmasını kontrol ettirdiğin "hurd" adını verdiğin objenin yatay kapsama alanı geniş olduğu için, karakterin o yatay alan bitene kadar havada süzülmesi gerekir. Çözüm olarak iki şey yapabilirsin; ya karakterin havada kalma süresini uzatacaksın, ya da objenin yatay genişliğini daraltacaksın. Perspektifsel değil de üstten düz bir hat gibi görünebilir. Ha bir de ben oyunun bu kısmını videoda felan görememiştim, ana sitesinde de fark edemedim 🙂
December 26, 2014 at 9:24pm · Like · 2

T2_S2 Bir de şey var, photoshop'ta uğraşmana gerek yoktu karakter hareketi için. Akademik 'ç'alıntı metoduyla halledebilirdin 🙂 http://dumbwaystodie.com/images/characters/toasty.png mesela çarpılarak öldüğü sahnenin kendi sitelerinde yer alan png serisi 😉

Figure 4.2 Helping each other without encouragement

As it can be seen from figure 4.3, discussion environment could easily be created with a little encouragement. In this screenshot, instructor assigned some of the students who have enough knowledge to solve the asked problem. It is important to select an environment commonly used by students as it was stated before. In this case, since the students are already active on the Facebook environment, the help of the students or instructor would be faster than other environments. As it can be seen from the screenshot below, after assigning students, the problem of the student was solved around 3 hours and solved completely in less than 7 hours. Students could help each other to solve their problems with the right medium and management. They can offer solutions for the errors they have encountered and provide instant help for each other. As it can be seen from the screenshot of the discussion, Additionally, the informal structure of the group helped the discussions to be warmer and friendly rather than being formal and homework-like which would be discussed further in the next code.


T2_S3 uploaded a file.
November 28, 2014

hocam bu sefer olmadı.. yarın tekrar yapıcam.

hareket.aia
File
Download

Like Comment

T2_S6 ✓ Seen by everyone

Instructor Ey T2_S2 T2_S6 T2_S4 bu sorunun yardımcıları olarak sizi atadim. Yardımcı olunuz
November 28, 2014 at 8:36am · Like · 2

T2_S4 Screen ayarlarından AlignHorizontal'ı "center " yapman gerek ki resim sol taraftan çıkıp gidiyomuş gibi gözükebilsin. Tabii sonrasında resmin x koordinatının ayarlarını tekrar yapman gerekecek.
November 28, 2014 at 11:46am · Like · 2

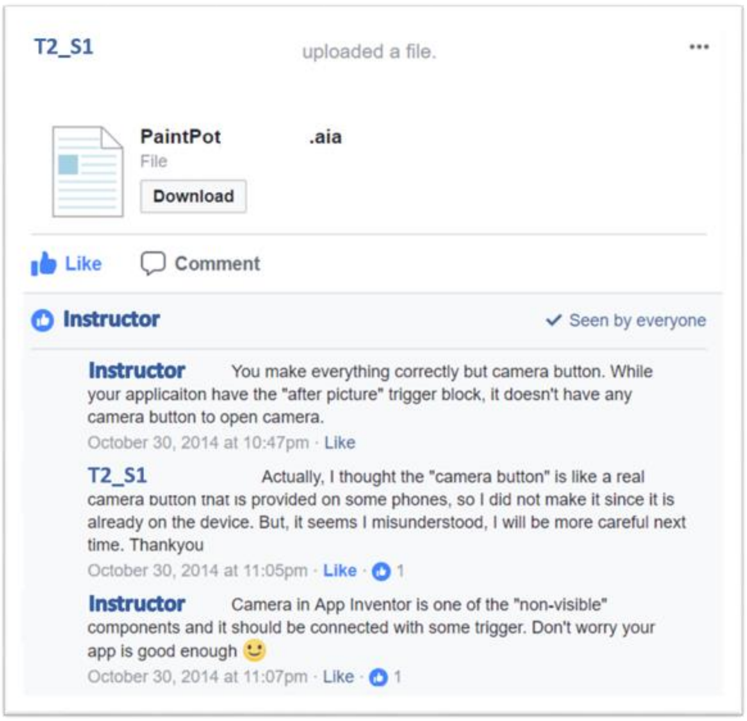
T2_S2 Aynen T2_S4 söylediği kısım doğru, bir de küçük bir detay Screen-Width sayısal değer olarak kullanacağın resmin (Width Değeri)x3 olarak ayarlaman işini kolaylaştırır. Resim olarak tercihin de süreğenliği sağlayıcı birşey olursa gerçekçiliği artar. Şöyleki resmin başlangıç kısmı bitiş kısmına ne kadar yakın olursa o kadar çok süreğenlilik ve gerçekçilik sağlanır.
November 28, 2014 at 1:18pm · Like · 2

T2_S3 dediklerinizi yaptım fakat şöyle bir sorunum var. resim 2i kez geliyor ekran bitince birdaha gelmiyor. ve yaptığımız şey çok basit fekat 50. kez deniyorum oluyor kafayı yiyeceğim az kaldı
November 28, 2014 at 2:44pm · Like · 1

T2_S2 Birinci resim ekrandan tamamen çıktığı anda diğer resmin sonuna taşıman gerekiyor. Genel süreci nasıl ayarladın bilmiyorum ama atıyorum resmin boyutu 300px'dir - canvas boyutun 900'dür ve onar px ilerletiyorsan, hareket halindeki resmin X'i 0 olduğunda... See More
November 28, 2014 at 2:53pm · Like · 2

Figure 4.3 Discussion environment to help each other

Facebook medium was also used to give feedback to students about their homework and projects. As it can be seen from the figure 4.4 below as an example, uploaded homework of a student was evaluated and been given feedback by the instructor. Students stated that he misunderstood the function of a block and after the feedback he corrected his mistake. Opportunities that medium provides could be essential for keeping students motivated and active towards the course.


T2_S1 uploaded a file.

PaintPot .aia
File
Download

Instructor ✓ Seen by everyone

Instructor You make everything correctly but camera button. While your applicaiton have the "after picture" trigger block, it doesn't have any camera button to open camera.
October 30, 2014 at 10:47pm · Like

T2_S1 Actually, I thought the "camera button" is like a real camera button that is provided on some phones, so I did not make it since it is already on the device. But, it seems I misunderstood, I will be more careful next time. Thankyou
October 30, 2014 at 11:05pm · Like · 1

Instructor Camera in App Inventor is one of the "non-visible" components and it should be connected with some trigger. Don't worry your app is good enough 🙂
October 30, 2014 at 11:07pm · Like · 1

Figure 4.4 Feedback from and Evaluation of Instructor

According to interviews and observations, students prefer Facebook to e-mail because of some of the additional features Facebook has. Rather than one-way communication, it has multi-directional communication. Commenting on an announcement or post makes them feel free. A more interactive, open, multi-directional and informal communication medium is what students seek for the communication of course.

Communication Medium as a Resource Hub

Another significant finding that students pointed out regarding the Facebook Group was that they also see and use the Facebook Group as a resource-sharing hub. While the resources utilized in the course like tutorials, links, etc. were shared in the web page of the course, announcements, questions about the homework of the students, progress of their projects were shared in the Facebook group. They use the group to overcome their mistakes in their homework or projects. In addition to the questions and shared problems, students also used old questions or posts to overcome their problems. As it was mentioned before, the openness of the environment also transforms the communication medium into a resource hub for students who are having difficulties. Student T1_S1 specified that while he was doing his project, he checked the posts of other students to draw a path for his project.

> We can see the posts. We can also use it as a resource when we were stuck. When I am working on the project, I looked there to see what my friends did and how they did theirs. (T1_S1)
>
> *Paylaşımları görebiliyorduk. Orayı aynı zamanda kaynak olarak da kullanabiliyorduk takıldığımız yerlerde falan. Ben proje ödevini yaparken bakmıştım, işte diğer arkadaşlar ne yapmış nasıl yapmış. (T1_S1)*

Student T1_S3 thought that the Facebook group was very beneficial to close the gaps in the topics that they have learned in class. According to the student T1_S5, they relied on each other as well as the instructor, when they have encountered a problem. Using the Facebook group to ask questions and help each other, remove the dependence solely on the instructor.

> Actually, I can say it was very beneficial to close the gaps. (T1_S3)
>
> *Hatta bayağı bir faydalı oldu diyebilirim. Açıkları kapatmak için. (T1_S3)*
>
> It was nice that everyone was asking questions or sharing something with each other. Because if my friend knows better than I do, about some point where I have some deficiency, s/he can help me. I mean we are not bound only to you. (T1_S5)
>
> *Facebookta herkesin birbirine soru sorması veya bir şeyler paylaşılması güzel bir şey, çünkü benim eksik olduğum bir noktada arkadaşım benden daha çok bi şey biliyosa o bana yardımcı olabiliyor yani sadece size bağımlı değiliz yani o noktada. (T1_S5)*

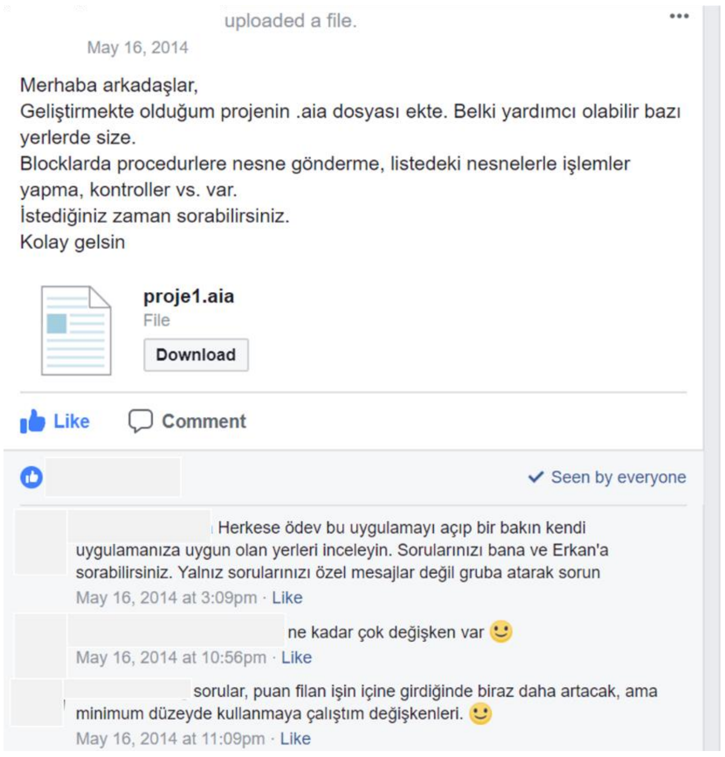
Encouraging students to share any problem they have encountered could help all of the students to learn from each other’s mistakes. Moreover, just seeing what others were doing in their project was also beneficial to help students to enrich their experience about developing their application. In this way, students with more knowledge about programming can share their experience and skills with first-time learners. Student T1_S8 emphasized that he benefited from the experience and mistakes of other students by using and looking at Facebook group. Similarly, T2_S6 also found the openness helpful for their learning.

> Using Facebook was seriously providing much help and it was very good. Because we learned some things from the mistakes and questions of our friends. (T1_S8)
>
> *Facebook'u kullanmak cidden çok kolaylık sağladı ve çok güzel oldu. Çünkü diğer sınıf arkadaşlarımızın hataları ve sorularında da biz bişeyler öğrendik orda. (T1_S8)*
>
> We have a chance to see what others did, not directly but more like how did he do it, what did he use. (T2_S6)
>
> *Diğerlerinin yaptıklarını görme şansımız oluyor orda. Direkt şey olarak değil de nasıl yapmış. Ne yapmış neyi kullanmış. (T2_S6)*

Student T1_S10 has also appreciated the open and transparent structure of the Facebook group. In addition to the capability of the medium, she also thought that encouragement of the instructor was also important to keep communication open and transparent.

> Seeing what everybody does was nice, indeed. So that, having it (Facebook group) was good, and you, saying, 'write there (group), do not send a message' was very good. (T1_S10)
>
> *Herkesin ne yaptığını görüyor olmak güzel tabi. Bu yüzden iyi oldu grubun olması sizin de hani daha çok mesaj değil de her şeyi oraya yazın demeniz gerçekten çok iyi oldu. (T1_S10)*

According to the observation notes, some of the students recalled the last week's examples to complete the tutorial. Retention of knowledge was observed by the instructor. After that week, instructor kept reminding the older examples to gave them hints from the tutorials of previous weeks. In addition to this strategy, it could be better to link the new knowledge with the old one at the start of the course and highlight the similarities and differences. According to observation notes, students also used the tutorial files as a resource to solve the current problem. Keeping all of the files accessible at all times and easy to access could help students to practically reach the information they need at any time they need. Other than checking the previous tutorials students also used projects of their classmates as a resource. Figure 4.5 shows that one student who was more experienced regarding programming than others shared the source file of his project to help his classmates. While one of them replied to the post, observations revealed that all of the students examined the project to overcome a problem in their projects.


[redacted] uploaded a file.
May 16, 2014

Merhaba arkadaşlar,
Geliştirmekte olduğum projenin .aia dosyası ekte. Belki yardımcı olabilir bazı yerlerde size.
Blocklarda procedurlere nesne gönderme, listedeki nesnelerle işlemler yapma, kontroller vs. var.
İstediğiniz zaman sorabilirsiniz.
Kolay gelsin

proje1.aia
File
Download

Like Comment

[redacted] Seen by everyone

[redacted] Herkese ödev bu uygulamayı açıp bir bakın kendi uygulamanıza uygun olan yerleri inceleyin. Sorularınızı bana ve Erkan'a sorabilirsiniz. Yalnız sorularınızı özel mesajlar değil gruba atarak sorun
May 16, 2014 at 3:09pm · Like

[redacted] ne kadar çok değişken var 🙂
May 16, 2014 at 10:56pm · Like

[redacted] sorular, puan filan işin içine girdiğinde biraz daha artacak, ama minimum düzeyde kullanmaya çalıştım değişkenleri. 🙂
May 16, 2014 at 11:09pm · Like

Figure 4.5 Using each other's Projects as Resource

4.1.2.2 Negative Sides of the Communication Medium

The researcher also asked students about the negative sides of the course regarding communication and Facebook group. Most of the students think that there was not any problem about the communication. The researcher also asked about a problem he encountered at the beginning of the course to reveal the reason behind this issue: Direct Messaging. Another possible problem was one communication medium could not be suitable for all of the students, especially in large classes.

Direct Messaging

One of the problems that instructor encountered at the beginning of the course was instead of sharing a question or a problem on the wall of the Facebook group, open to everyone; some of the students asked their questions to the instructor or their friends through the direct messaging. Throughout the course, Instructor encouraged the student to share anything they have had trouble with on the “wall” of the Facebook group. If they have a question, the instructor told them to ask that through posts on the wall. In this way, the students will see each other’s problem and learn from the other’s experiences, mistakes and difficulties. Even if they do not have any difficulty, they ought to share the progress of their project. Researcher aimed to create a more open and transparent communication group, in which students can improve their knowledge by following the Facebook Group. One problem about the Facebook group was that some of the students used direct messaging instead of sharing. When the direct messages from students on the Facebook group were examined, students who had difficulty while developing their applications used direct messaging. The Interviews and examination of the documents revealed that students not only sent messages to the instructor, but also to their friends. The researcher investigated this issue to see the cause; the main reason was that they were shy as students T1_S11 and T2_S1 put forward. Since they are new to the domain, they think that their question is too easy to ask and their classmates will laugh at them.

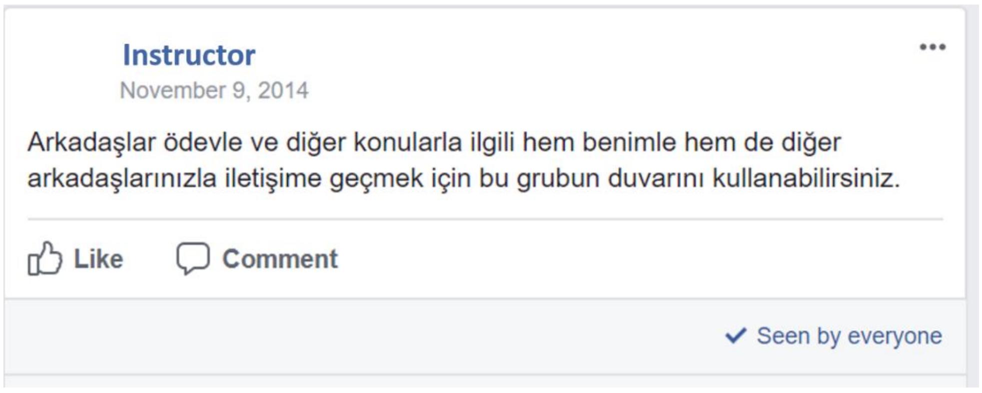
> We are shy. (T2_S1).
>
> I mean, If it was a very simple thing that I did not understand, I mean not to be in front of my friends’ eyes. I mean, it is a psychological thing actually. (T1_S11)
>
> *Hani böyle çok basit bir şeyse anlamadığım hani diğer arkadaşların gözü önünde şey olmaması için hani bu biraz psikolojik bir şey aslında. (T1_S11)*

Students T1_S5 were thinking that their questions/problems were too easy to ask and they were ashamed to ask open to everyone.

> For example, there is something so simple, and you look at it like “is that even a question to ask?” People sent direct messages… I mean the reason behind this ‘would our friends laugh at me?’ You know, kids in elementary schools afraid of raising their hands, because it is an easy question. It was like that. (T1_S5)

Mesela çok basit bir şey oluyo ya bu da sorulur mu gibisinden bakıyorsunuz. Özelden mesaj atılıyor… Yani işte arkadaşlar bize güler mi diye bi mantık. İlkokuldaki çocuklar gibi parmak kaldırmaya korkan çocuklar oluyor kolay bi soru diye. Onun gibi. (T1_S5)

Constant encouragement could be needed to get rid of this barrier. Instructor reminded students to share all of their questions and problems through the wall of the Facebook group. As it can be seen from the figure 4.6, instructor also used the communication medium to remind and encourage students to share their questions and problems through the Facebook wall. This problem was solved after the encouragement of the instructor. Instructor also made it mandatory to share the progress of their final project regularly. Students shared the progress of their project with screenshots from design and blocks screen, once every three days. After the encouragement of the instructor and other students answered their questions instantly, students also feel relaxed about sharing their problems.



Instructor

November 9, 2014

Arkadaşlar ödevle ve diğer konularla ilgili hem benimle hem de diğer arkadaşlarınızla iletişime geçmek için bu grubun duvarını kullanabilirsiniz.

Figure 4.6 Providing Encouragement through Communication Medium

Using a Single Communication Medium

Although nearly all of the students liked and preferred using the Facebook, not all of the features of Facebook were interesting for everyone. One of the students stated that he only used Facebook for their courses. This showed that if there was not a consensus about the communication medium of the course, instructors could use more than one medium or support the main medium with another. The student was checking the updates from his e-mail account and login Facebook to help others or check the homework posts. The student was very helpful to his friends when they have a problem with their project or homework. Without being an active user, he could not see the posts, if Facebook did not have an e-mail notification feature. So, the instructor should make sure that every student was kept updated.

> I mean, I don’t like to use it, so… I checked the posts from my mail. If there was something I need to write, I was logging in and wrote it. (T2_S2)
>
> *Yani ben de çok kullanmayı sevmediğim için. Oradaki mesajlaşmaları genelde mail üzerinden kontrol ediyordum. Yazmam gereken bir şey varsa girip yazıyordum. (T2_S2)*

Other than this exception communication medium was preferable for the students, however in classes with higher number of students, it could be harder to reach a consensus. Alternative media could be used to support the students who do not actively use the medium chosen by the majority.

4.1.3 Summary of the Communication Theme

As the overview of this theme, some of the crucial points have been emphasized which could help the design of new introductory programming courses. Some of the findings are also eligible for courses from different domains.

- Constant and strong communication with instructor helps students to eliminate difficulties of the course
- Informal and sincere communication could affect the attitude towards the course and the topic
- Choosing an appropriate communication medium based on the opinions of the students which
    - Is commonly used and popular among students
    - Allows instant communication between students
    - Is convenient and practical
    - Allows open and multi-directional communication
- Using communication medium not just for communication and announcements but also as a hub for resources for students which put the student into the center and allow them to
    - Learn from each other
    - Track each other's projects

  - Ask questions about their problems to instructor or peers
  - Check out the old problems and examples

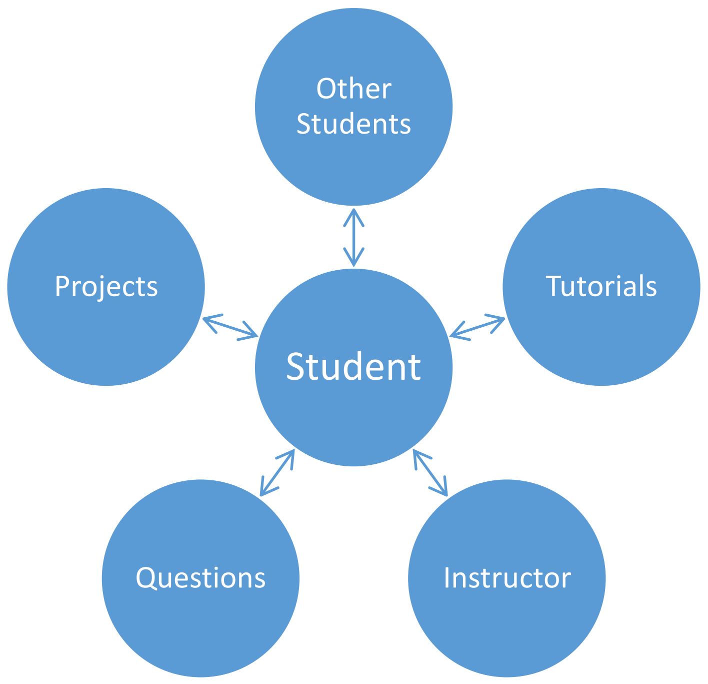


Figure 4.7 Using Communication as a Resource Hub

- Encouraging students to share their problems and questions open to everyone which will help students to learn from each other's mistakes and questions
- Encouraging peer help and canalizing students to help each other could also help to reduce the workload of instructor and prevent instructor overload

4.2 Contributions of the Course

Contributions of the course to the students emerged as another theme from the interviews and observations.

4.2.1 Transfer/Link to Professional Life

Students were asked if they have a plan in the future regarding App Inventor. In addition, the researcher asked them if they have a plan to develop an application in the future. There were two answers come forward among them. Most of the student emphasized that they will develop an application if they need an application. Need is what motivates them to continue to use the knowledge, they have acquired. Another popular answer was career-wise. If students have a career related to programming, they stated that they are going to use their knowledge to develop an application or teach others what they have learned in the course. And the students from other fields also thinks that learning programming is beneficial for their career, regarding both material development, understanding the programming logic and relating their field to programming.

4.2.1.1 Teaching Programming

Students planning to use the programming knowledge they acquired from the course, show that they have a clear understanding of programming logic. Some of the students who were taking the course were from the Department of Computer Education and Instructional Technology. Main job opportunity for these students is becoming information technology teacher. They were willing to use the App Inventor environment to teach their students programming logic as T2_S5 stated.

> Yes, I feel like I can use it when I teach programming logic. (T2_S5)
>
> *Evet kullanabilirim gibi geliyor bana. Programlama mantığı vereceğim zaman. (T2_S5)*

One student (T1_S11) who was willing to be a mathematics teacher in future, thought that developing applications would increase the motivation of the students. She also wants to teach the App Inventor to help them understand that math and programming are in direct relation. She also stated that she has already some students who are willing to be a computer engineer but do not like the mathematics. So, she wanted to use his knowledge of programming to show the relationship between his field and programming.

> I mean, I can make something to increase the motivation of the children. Mathematics is necessary, to develop an application. I can show the application and say ‘do this like this’ I mean you have to use the math. (T1_S11)
>
> *Yani çocuklara hani şey yapabilirim onların motivasyonunu arttırmak için hani. Matematik şart mesela uygulamayı yapmak için. Uygulamayı gösterebilirim bunu böyle yap hani matematik kullanmanız gerekiyor. (T1_S11)*

Another student T2_S4 who was a student in the business administration program thought that knowing programming could be beneficial for his career. Especially if he would work for a technology company.

> I think that if I work for a technology company, I mean knowing the subject, even if you are a supervisor or the owner… It is beneficial for understanding what employees do, or for helping them, or for supervising them. (T2_S4)
>
> *Teknoloji şirketinde çalışsam diye düşünüyorum, yani konuyu bilmek, yönetici olsa bile veya başka bir işletmeci olsan bile hani çalışanların ne yaptığını bilmek ya dayardımcı olmak açısından ya da denetlemek açısından bence sağlar. (T2_S4)*

4.2.1.2 Product Development

Developing an application after the course is over, could be an important indicator for both motivation and knowledge about programming. Interviews revealed that 15 of 18 students are planning to develop applications for business or leisure purposes.

Supporting the Career

They also tend to use their knowledge to develop their application for their professional life. Most of the students would not/could not choose programming as their main career. However, they think they can use their programming knowledge to support their career or develop their own application to help their students or co-workers. Student T1_S2 feel competent enough to develop his own application, and since his plan is becoming a mathematics teacher, he thinks that he can develop applications about education.

> For example, we can develop applications now, or at least if we work on it we can make something out. We develop one on our own, if we want, we can develop something about education. It could be for material (development)… I don't know… In charter schools, teachers share applications with their students. Teacher installs it on the tablets. I mean you can develop simple things like those on your own, why not. (T1_S2)

> *Mesela uygulama geliştirebiliyoruz ya en azından artık üzerine uğraşırsak bir şeyler ortaya çıkarabiliriz. Bir tane uygulama kendimiz geliştirdik istersek eğitimle ilgili bir şeyler geliştirebiliriz. Materyal olabilir. Ne bileyim… Özel okullarda falan hani uygulama paylaşıyormuş hocalar. Hoca koyuyo tabletlere gidiyor. Hani o tarz bir şeyler kendin basitçe bir şeyler yapılabilir, neden yapılmasın. (T1_S2)*

Student T1_S6 and T1_S7 are students of the elementary science education department, and they both think that developing applications for their future students will provide a more effective teaching opportunity for them and fun learning for students.

> Yes if I can do it, it would be very good. For example, this one (mentioning the project) was about my field of expertise. It would be nicer to improve it, present it to students, and teach them basic stuff in a more entertaining way. (T1_S6)

> *Evet yapabilirsem çok iyi olur mesela bu yaptığım benim alanıma yönelikti ya. Onu daha gelişmiş bir şekilde yapıp, öğrencilere sunup, onlara hem eğlenceli şekilde vermek falan güzel olurdu yani basic şeyleri. (T1_S6)*

> I can make them (his students) understand a topic better. I mean, maybe I can make it more entertaining by using games. (T1_S7)

> *Bir konu hakkında daha iyi anlamalarını sağlayabilirim belki hani oyun şeklinde hani daha zevkli hale getirebilirim. (T1_S7)*

Student T2_S6 also tends to use his knowledge acquired from this course for his professional life. He has thought about developing an application for the law office he was working the time he was interviewed.

> I can develop simple programs directly related with business life. It can be for our office. (T2_S6)

> *Bir de direkt iş hayatına yönelik basit programlar şey yapabilirim. Bizim ofise yönelik de olabilir. (T2_S6)*

On the other hand, based on the observations, student T1_S9 who was an EME student was planning to develop an application to make money. Even though his education was teaching related, he can use his knowledge of programming to switch careers. As his final project, he made two applications, even if one was enough. He made a breakout clone using bonibon candies (similar to M&M candies) rather than bricks. One of his plans was selling the game to the company which makes the candies.

> Yes, I have. I have a thought that putting it on Play Store and being rich. (T1_S9)
>
> *Var hocam evet. Play store'a koyup zengin olma gibi bir düşüncem var. (T1_S9)*

Observation during the course showed that relating the knowledge to their or to a career towards programming career is a strong motivator for the students. Observation revealed another contribution of the course: students constructed the idea of developing an application they need.

Need/Leisure Time/Entertainment

Interviews revealed that it is important for students to have the ability to develop an application by themselves any time they need. Even if they did not want to be a software developer, or they did not want to use it in their professional life, students still think that what they have learned provide them the belief and opportunity of developing any application they need. While some students mentioned before are planning to use the skills they have from the course, some of the students are planning to develop applications based on their needs, or just for entertainment.

> I took a step like this, so it was beneficial for me. I can do little apps with App Inventor that will come to my mind. (T1_S4)
>
> *Böyle bir adım atmış oldum benim için de çok faydalı oldu. Küçük aklıma gelen şeyleri yine App Inventor ile yapabilirim. (T1_S4)*
>
> If I need anything, I will try at least. (T1_S6)
>
> *İhtiyaç duyduğum bir şey olursa denerim en azından. (T1_S6)*

Yes, I will use most likely. I mean, I am going to work on the list creation. I am going to learn that one… Exclusive to me… (T1_S10)

Evet yani kullanırım büyük ihtimalle. Yani bu liste oluşturma üzerine zaten uğraşıcam öğrenicem onu… Kendime özel… (T1_S10)

Student T1_S2 states that he would prefer to create his own application before he would look for it in Play Store. He thinks that he would put some effort into it before using a prepared one.

I mean, if I need something rather than searching the Google Play, I think I can make something by myself. Even if I could struggle a little, I think I can manage to get my work done. (T1_S2)

Yani bir ihtiyacım olduğunda direkt Google playden aramak yerine bir şeyler yapabilirim diye düşünüyorum. Biraz uğraşabilirim basit de olsa işimi halledecek bir şeyler olacak diye düşünüyorum. (T1_S2)

Student T1_S5 exemplified one of the benefits of App Inventor knowledge as developing applications for the children relatives of his.

It will provide benefits because I have a lot of children relative… Making something for them would be nice. (T1_S5)

Yarar sağlar çünkü bayağı bir akrabam var küçük çocuklardan. Onlar için bir şey yapmak güzel olur. (T1_S5)

Student T1_S11 emphasized the need as the other students did, and she thought that she could use it to improve her abilities in programming to develop an application.

In case there is a need for anything, it is something I would want to do, or I could want to improve myself, I don't know… Then I can research and make something in App Inventor. (T1_S11)

Herhangi bir şey için hani ihtiyaç halinde yapabileceğim bi şey ya da kendimi geliştirmek isteyebilirim bilmiyorum. Hani şunu da

> *kullanayım diye o zaman araştırıp bir şeyler yapabilirim App Inventorda. (T1_S11)*

Some students think that they can work in the software development field with the help of what they have learned from the course. Students T1_S8 who is a CEIT student planning to develop an application about psychology.

> Yes. We are thinking about a project with my brother. We are thinking about developing something related to psychology. (T1_S8)
>
> *Evet. Bunun için yani kardeşimle aslında ortak bi proje düşünüyoruz... Psikoloji ve psikolojiyle alakalı bir şey geliştirmeyi düşünüyoruz. (T1_S8)*

Another CEIT student, T2_S3 was thinking that if her career will move towards mobile application development, or if she needs to develop one, she would develop an application.

> I do not have anything in mind, but I took the course thinking, “maybe android programming could help me in my career in future” Now after finishing the course, improving ourselves is up to us. Of course, if I start to work in that business area, of course, I will be able to develop an application. (T2_S3)
>
> *Aklımda bir şey yok ama ileride belki bir iş dalında android programlama bana yardımcı olur düşüncesi ile dersi almıştım. Şimdi dersi aldıktan sonra bunu geliştirmek üstüne koymak artık bizim elimizde. Tabi ki de ileride bir iş alanına girersem ya da bir şeye ihtiyaç duyarsam tabi ki de bir program geliştirebilirim. (T2_S3)*

Student T2_S5 was planning to develop an application to meet her need regarding her hobby.

> I am thinking (to develop) about tv series follower app. Like, I will enter the dates of the shows I watch, at the beginning of the season. When I touch the one I watched, it will disappear. (T2_S5)

> *Dizi takip şeyi gibi böyle düşünüyorum. Kendi izlediğin dizileri falan tarihleri falan dönem başında giricem. İzlediğime tıklayacam o gidecek. En azından, ben de kaldığım yeri bilecem. (T2_S5)*

Observations during the course hours supported the code revealed after the interviews. Students constantly had the ideas of developing a program based on their needs. One of the students (T1_S9) even developed a delivery tracking application depending on his need to track his orders from different online shopping retailers. At the beginning of the second week, another student came up with an application idea that he needed. Student asked the instructor what he needs to develop that application which had to be developed using proximity sensor and was not available at that time in App Inventor. The student had to give up that idea because of the constraints of App Inventor. Throughout the course “the need and entertainment” was one of the essential stimulants for students to go further about programming. Students constantly generated ideas about their possible projects, after learning new topics and components.

4.2.1.3 Using AI for Other Courses

Students were also asked whether they could use the App Inventor for other courses. While some students say that they can use their knowledge about programming in other courses, they are planning to take in future; others could not make a connection between them. Student T1_S4 suggested App Inventor as an alternative programming environment to be used in other courses. He thought that it would be better to use App Inventor instead of more commonly used Adobe Flash as the development environment.

> It would exceedingly be beneficial for the other courses. In some courses, we still use Flash. Instead of that, if we can develop a game or application as the project of that course, I believe that it would be far more beneficial. (T1_S4)

> *Diğer dersler için yarar sağlar fazlasıyla bazı derslerde hala flash yapıyoruz onun yerine app inventorda bi tane oyun ya da uygulama yapsak o proje, o dersin projesi kapsamında çok daha faydalı olacağına inanıyorum. (T1_S4)*

Some of the non-majors of computer area were also thinking that using App Inventor could help their courses. Student T1_S10, an EME student, believed that that App Inventor could be used to support and concretize the subjects in their courses could be beneficial for them.

> I mean my classes mostly based on theorems. Related to that subject, I want to develop an application that collects all theorems in one place… I mean I want to create a program like that. (T1_S10)
>
> *Ondan sonra derslerimi hani zaten çok teoremlerden oluşan bir ders onlarla ilgili hani bir konuyla alakalı tüm teoremleri bir yerde toplayıp hani o şekilde bir program yapmak istiyorum. (T1_S10)*

Student T2_S7 who was also an EME student found App Inventor useful to create materials for her courses.

> We develop (teaching) materials a lot. This can be beneficial in the second semester while developing those. We make a lot of games, PowerPoint (presentation) and stuff. It would be easier. (T2_S7)
>
> *Mesela şeyle alakalı olabilir biz material falan çok hazırlıyoruz. 2. Donem onları hazırlarken işime yarayabilir. Oyunlar falan çok yapıyoruz PowerPoint şudur budur. Daha kolay olur. (T2_S7)*

Observations also revealed that students saw the App Inventor as a beneficial developing environment for other courses. As student T2_S7 put forward during the interviews, students from elementary mathematics education in the first semester also asked during the course hours, if the App Inventor environment would be a good alternative to PowerPoint to develop educational materials with more interaction.

4.2.2 Computational Thinking

4.2.2.1 Definition/Characteristics of Computational Thinking

Students did not know about the concept of computational thinking as they were focused on programming solely. However, interviews revealed that they have some idea about computational thinking with different names or definitions. Even though they were not aware of the concept wholly, they were familiar that programming has a different kind of logic. According to the observation notes from the first week of the first term, when instructor asked the student their opinions about the first week, they stated that it was very good. And when instructor added that next week programming logic will be more pellucid, they were happier. That means students were expecting and motivated to learn programming and its logic even though they do not know the characteristics fully but they were aware.

One of the students (T2_S2) from CEIT department expressed his opinion of the benefits of this course to the new learners. He named the benefits as mathematical thinking and strategical thinking which are the concepts both related to computational thinking ability.

> It has two things that could provide benefit. First, it can be a preparation for programming courses. Plus, mathematical thinking is in the course so let's say, you are going to give a motion (to an imagesprite) you are thinking about coordinates. You will add an object, again "where to put it in coordinates." When you are creating a motion how much to increase or how much to decrease, I mean it is nested inside mathematics a lot, so it develops mathematical skills too. Other than that, not as a course, it provides strategical development. I mean it improves strategy of the person, or I can say strategical thinking. (T2_S2)
>
> *Yarar sağlayabileceği iki şey var zaten birisi programlama derslerine ön hazırlık gibi bir şey olur. Artı matematiksel düşünce de işin içinde olduğu için sürekli atıyorum bir hareket verecekseniz koordinatları düşünüyorsunuz. Bir obje koyacaksanız yine koordinatlar üzerinde nereye koysam. Hareket sağlarken ne kadar arttırayım ne kadar azaltayım yani bayağı bir matematikle iç içe olduğu için matematiği de geliştirmiş oluyor. Onun haricinde ders olarak değil de strateji geliştirmeyi de sağlıyor bir şekilde yani stratejiyi de geliştiriyor kişinin. Stratejik düşünmesini diyeyim ya da. (T2_S2)*

Another student who was an elementary mathematics education student (T1_S11) also related the mathematics to programming. She connected the logical reasoning in mathematics with computational thinking.

> Also, there is a thing called logical reasoning. We use it in this course. Source of it is actually mathematics. (T1_S11)
>
> *Bir de mantıksal akıl yürütme diye bişey var hani. Onu bu derste hani kullanıyoruz. Onun da kaynağı aslında matematik. (T1_S11)*

The same student was thinking that, after taking the course, the algorithmic thinking approach changed his way of thinking. Moreover, she expressed some of the characteristics of computational thinking in the literature (Grover & Pea, 2013; Wing, 2006) without knowing the concept.

> At least, I can say that it helps us to think differently. I mean, as a guide, to reach the end, I can say that it helped me to conclude it is needed to pass through those paths. You break it into pieces and reach to the whole. (T1_S11)
>
> *En azından farklı düşünmemizi sağlıyor diyebilirim hani bi yol gösterici hani bir sona gitmek için bu yollardan geçilmesi gerektiği gibi bi sonuca varmamı sağladı diyebilirim. Parçalara ayırıp sonra bir bütüne gidiliyordu. (T1_S11)*

Similar to student T1_S11, during the interviews 7 more students stated that their thinking styles about their daily routines were changed which will be mentioned in the following code.

4.2.2.2 Real Life Examples/Realization

Even the simple connections to their daily routines helped them think differently, and change their perspective and way of thinking. In their daily lives, students realized that they already were using the algorithmic thinking. At the beginning of the courses, students were asked to write the steps of how to brew tea. They found out that they make their daily routines by dividing them into little steps. After the realization that thinking style got more explicit for them. Integrating a new thinking style to solve the problems, not just for programming course but also for their everyday activities, could help them to integrate computational thinking as their living style. After the very basic examples, student T1_S2 stated that learning the basic algorithm examples changed her perspective

> After the first class, my point of view has changed towards daily life. You know, you told us to brew tea. I mean, I started to think in that style, wondering how that will do. (T1_S2)
>
> *İlk dersten sonra bakış açım değişti gündelik hayata bazen. Hani siz dediniz ya çay demle. Hani, o tarz şeyleri biraz düşünmeye başladım böyle acaba nasıl olur falan diye. (T1_S2)*
>
> You know how to brew a pot of tea but dividing it into step by step adds you something new. You can implement this to any field actually… No one says 'I have this problem, I will write the algorithm of this' but when he was thinking, at least, they come to his mind. (T2_S6)
>
> *Çay demlemeyi biliyorsun ama onu bir de basamak basamak yazmak adımlara bölmek yeni şeyler katıyor. Her alanda uygulanabilecek bir şey aslında. Kimse şey yapmaz hani benim şöyle bir problemim var onun hadi algoritmasını yazayım demez ama en azından düşünürken şeyler gelir. (T2_S6)*

Students thought that they realized that they were already using the algorithm in their minds, but this course made them realize and create awareness about their thinking style. Students stated that they noticed real life, daily routine examples leads to explicit the algorithm concept.

> The things I passed by superficially have become more attention-grabbing. (T1_S2)
>
> *Böyle üstün körü geçtiğimiz şeyler şimdi dikkat çekici olmaya başladı. (T1_S2)*
>
> We had been doing some things related to the algorithm, but we were not aware of it. We had been hearing from computer engineers (students) like 'this is my algorithm' and stuff. This adds something to us. (T1_S3)

> *Daha önce yapıyormuşuz* algoritmayla *ilgili bir şeyler ama onun farkında değildik hani algoritma üzerine. Bilgisayar mühendisliğinden falan duyuyoduk benim algoritmam falan diye. Onu işte eklemiş* oldu. *(T1_S3)*

> It creates awareness. You know how to brew a pot of tea, but you realize that it is an algorithm, and realize the decision mechanisms. (T2_S6)

> *Şey oluşturuyor farkındalık oluşturmuş oluyor. Çay demlemeyi biliyorsun ama bunun aslında algoritma olduğu karar mekanizmalarının falan farkına varıyorsun. (T2_S6)*

Even though student T1_S8 have taken a programming course before in the same department with a textual programming language (C++), he stated that after this course they have realized they have gained a skill that helps them to see the alternative solutions and choose the shorter, more practical one.

> For instance, you will develop a program. You have two ways; one is long other is short. This (course) taught me to choose the short one, I mean, it taught that short way. (T1_S8)

> *Mesela bir program yapacaksınız bunun iki yolu var bi uzun yolu var bi de kısa yolu var. Kısa yolunu seçmeyi öğretti yani o kısa yolu öğretti. (T1_S8)*

One of the students reported that programming means the translation of the human mind to the computer. He found programming interesting by its nature. Moreover, students discovered a new way of thinking; they realized that their minds divide big problems into little steps just as a computer does. They connect their mind with the computer’s working style. They also find this way of thinking useful.

> I like it because it is challenging how to translate from our mind to the computer. So it is interesting in itself. (T2_S1)

> For the introductory learning… I understood the programming logic now. I mean, this helped a lot to us to understand the programming logic. (T1_S2)

> *İlk öğrenim için… Programlamanın mantığını anladım ben şu anda. Hani ona çok faydası oldu bence programlama mantığını anlamamız açısından. (T1_S2)*
>
> Strategical thinking helps everything. (T2_S2)
>
> *Stratejik düşünce her şeye yardımcı olur. (T2_S2)*

In addition to computational thinking, according to the students, real-life examples make students to learn the fundamental concepts of programming better. T1_S8 stated that learning concepts like ‘if’ and ‘function’ through daily life example was better.

> In class, for instance, when we give examples of real, daily life, we understood it better. For example, we are learning “if”, “function” and stuff. We learned those from a daily life example. It was nicer with that method. It was like integrated with our lives. We had the chance of exemplifying it as it is so that we understood them better. (T1_S8)
>
> *Derste mesela örnek verirken günlük hayattan örnekler yapılıp daha iyi bi şekilde anlıyoduk olayı mesela. Mesela if leri öğreniyoruz fonksiyonları öğreniyoruz başka bir şeyi öğreniyoruz. Günlük hayattan bi örnekle öğreniyoduk bunu yani daha güzel oluyodu bu şekilde. Bizim hayatımızla da içiçe girmiş oluyodu. Birebir örnekleme yapma şansımız oluyordu daha iyi anlıyorduk. (T1_S8)*

Even though this course was their first time ever to see and try computer programming for seven of the eighteen participants, during the course they were highly motivated and curious about programming logic. According to the observation notes, in the first lab hour of the course, when the instructor mentioned that they will learn more about the programming logic, especially students with no programming experience were happy and enthusiastic. Motivation and enthusiasm of students should be encouraged and fed by the instructor regularly. Being aware of the different thinking style is an essential step to develop more complex applications. The instructor should give examples and reminders about computational thinking and its thinking style and strategies.

4.2.3 Learned Concepts of Programming

Students have also stated that they learned new concepts about programming. Students with no previous programming knowledge stated their general knowledge about the programming changed their perspective and they learned the programming logic. Learning programming in itself is important for students with no previous knowledge since programming was known as challenging and scaring topic to learn. Being more familiar with what programming could have changed the attitude of the student T1_S7.

> I, for instance, had no idea about programming, maybe it gave me some information about programming. It seemed to me that which part makes what was too complex, I mean I am more familiar now. (T1_S7)
>
> *Benim mesela programlama yönünde hiç bir bilgim yoktu belki programlama yönünde bana bilgi verdi. Neyin ne işe yaradığını çok kompleks geliyodu bana biraz hani ona aşina oldum. (T1_S7)*
>
> I mean, at least I have an idea about programming now. I didn't have a clue about what it is, or it is not. (T1_S6**)**
>
> *Yani, en azından bi fikrim olmuş oldu programlamaya dair. Hiçbir şey bilmiyordum nedir ne değildir. (T1_S6)*
>
> Certainly, it added something to me. My computer knowledge has developed even more. I can do things better now. (T2_S7)
>
> *Muhakkak bir şeyler kattı ya. Bilgisayar bilgim bir kere biraz daha gelişti. Bir şeyleri daha iyi yapar oldum. (T2_S7)*

Students with previous knowledge about programming gave some examples of what they have learned better with the help of this course. Student T1_S8 stated that he had taken a programming course, but some of the concepts like variables have gotten better after this course.

> I understood the variables better… They have gotten better. (T1_S8)
>
> *Şu variablesları daha iyi anladım. Onlar daha iyi oldu. (T1_S8)*

Even the students with programming knowledge stated that this course could be useful for the students in their department. They liked some of the concepts they have learned in this course which they have not in the other programming courses. One of them is the time concept. Clock component in App Inventor was both found difficult to understand by some students, but it was also seen as useful.

> Our friends who will graduate from our department should know this, in my opinion. (T1_S4)
>
> *Ama bizim bölümden mezun olan birçok arkadaşın bilmesi gerekiyor bence. (T1_S4)*
>
> Another thing is the time concept. I liked the clock very much because for programmers, I don't know programming so well but as far as I can see, there aren't any time concept in C, C++, Java. There are loops but there wasn't much regarding time, in what we learned. (T2_S6)
>
> *Bir ikincisi de zaman konusu çok şey yaptım clocklara çok hoşuma gitti çünkü bir programcılar için programlamayı tamam çok fazla bilmiyorum ama benim gördüğüm kadarıyla C de C++ da javada şunda bunda zaman kavramı şey yok döngüler var ama zaman olarak da çok fazla bir şey yok bizim gördüklerimizde. (T2_S6)*

4.2.4 Summary of the Contributions Theme

Students shared their opinions and experiences regarding the contributions of the course to them. Students emphasized some of the important parts as follows:

- To be able to develop an application makes students to
  - Feel competent
  - Feel motivated towards subject
  - Change their attitude towards programming and stop them to fear from programming
- "Need" was one of the primary sources to drive students to develop applications.
- Connecting their knowledge with their career and using the knowledge from the course to develop applications for their career

- According to students, to be able to teach programming by using App Inventor is another benefit for the career of students
- Using the knowledge for other courses with App Inventor is another contribution based on the interviews. The main use for other courses:
    - Alternative development environment for computer related courses
    - Rather than preparing passive presentations with PowerPoint, developing an interactive app
    - Support material to concretize the abstract concepts
- Another contribution was learning programming or concepts about programming
    - First-time learners stated that they overcame their fear about programming since they have learned the basic concepts
    - Students with previous knowledge indicated that they have learned difficult concepts better
- Activities in the course create change about their thinking styles
    - Realization: Real life algorithm examples make students to “realize” how they divide a procedure or a daily routine into little steps
    - Awareness about the algorithms and different thinking and solving process
    - Making their thinking styles explicit
- Students named some of the effects of this course to their thinking styles as they named it:
    - Change of perspective
    - Strategic thinking
    - Algorithmic thinking
    - Mathematical thinking
    - Logical reasoning
    - Programming logic
    - Translating mind to computer
    - Dividing problems into steps
    - Choosing the easier solution among alternatives

According to the findings under this theme, “Need” was what keep students to plan developing applications, after the course ended. Feeding the need with giving application ideas or uncover their daily problems which could be solved with an application. Another contribution was students’ using the knowledge for their careers. Transferring their knowledge to their daily or professional life is essential to help them to improve themselves. Throughout the course, linking the knowledge to the careers of the student could be an effective strategy both for motivation and effectiveness of the course even after the course has ended. Moreover, examples related to their educational field could also be an effective strategy to help them to see the connection between the other courses and programming to develop material for those courses. Students think that they can use App Inventor to develop interactive applications for their courses that could concretize the abstract topics.

Students were also thinking that this course gave them a new perspective and thinking style. Students use computational thinking in their daily lives without realizing. Turning a tacit knowledge into an explicit one could help students to be aware of the problem and practical way to solve it. According to students, giving real-life problems and examples assist them to see and evaluate the situation in a different way.

4.3 Motivation

Motivation could be one of the most important factors to help novice programmers to continue learning. In this theme, what motivates and what demotivates students were investigated. This course and the environment were related and very close to each other. Therefore both the environment and the design and strategies emerged from the course will be examined one by one.

4.3.1 Visual Environment

Visual environments hold potential to motivate the novice programmers towards programming. Comparing to the textual programming environment, visual programming environments have some advantages in general with the help of course implementation. While visual programming is not limited to App Inventor environment, the students also mentioned some special visual features of App Inventor environment. Student T1_S2 stated that using blocks instead of textual codes make the environment more attractive for the first time learner.

> When blocks are ready-to-use, and everything was hidden inside codes, the environment became a little more attractive. For the first time learning… (T1_S2)
>
> *Böyle bloklar hazır her şey kodlar içinde gizli öyle olunca biraz daha cezbedici ortam oldu. İlk öğrenim için… (T1_S2)*

Student T2_S3 listed some benefits of visual environment as having visual feedback, visual and concrete products.

> I like designing stuff. And also, there is visual feedback actually. I mean, maybe because we acquire concrete things… Even though I took the course as an elective course, I labor for this course the most among others, and I enjoyed this one the most. It is very good for it to be visual. (T2_S3)
>
> *Seviyorum, bir şeyler dizayn etmeyi. Bir de geri dönütü görsel olduğu için aslında. Yani daha somut şeyler elde ettiğimiz için belki de. Seçmeli olarak almama rağmen dersler arasında en çok uğraştığım en çok eğlendiğim derslerden biriydi. Yani görsel olması çok iyi. (T2_S3)*

Same student (T2_S3) who have previous programming experience states that having a visual product also motivates her to study. This shows that motivation of visual products are not limited to any environment including the visual environment.

> When I worked with Jquery, because I have something visual in my hands, I was working more enthusiastically, more willingly. If I explain this with the applications we make with App Inventor, frankly, I like it more because it is visual and I worked more enthusiastically. (T2_S3)
>
> *Jquery yaptığım zaman görsel bir şey olduğu için elimde ben daha çok daha hevesli çalışıyordum daha istekli çalışıyordum bunu app inventorla yaptığımız programlarla açıklarsam görsel açıdan olduğu için benim açıkçası daha çok hoşuma gitti daha hevesli çalıştım. (T2_S3)*

Student T1_S10 found C++ course as a theoretical course in which she did not understand what they were making because they did not see a product through the creation.

> That (C++ course) was a little theoretical; we did not see what we are making. It comes out when we compile but App Inventor is more enjoyable, we can see directly as an application. We can see it right on the phone. (T1_S10)
>
> *O biraz daha teorik kalıyodu aslında görmüyoduk ne yaptığımız pek aslında yazdırınca çıkıyodu ama App Inventor tabi daha zevkli direkt uygulama olarak görebiliyoruz. Telefonda falan direkt şey yapıp görebiliyoruz. (T1_S10)*

As the easier environment was one of the important motivators for student regarding the course, it also motivates students to develop their own applications in future.

> I mean, I can make something that I need, for example. We made the Google search engine example. I was surprised when I saw that. I showed that to my friends, and they were very surprised. It seems like it is very difficult for them, but it was easy. (T1_S7)
>
> *Yani ihtiyacım olduğu bir şeyi yapabilirim mesela. Şeyi Google gibi arama motorları yapmıştık bir tane onu gördüğümde çok şaşırmıştım. Arkadaşlarıma gösterdim falan da nasıl yaptın falan diye çok şaşırıyorlardı. Çok zor bir şeymiş gibi geliyodu onlara ama kolaymış. (T1_S7)*

4.3.2 Non-intimidating course design

Course’s non-intimidating design was also another source of motivation for the students. Since the course was an introductory course and the subject was intimidating for the most of the learners, the course and the environment should not be too difficult for the students. Moreover, the instructor should make sure that examples are relevant, and easy for every participant, and open to development for the fast learners. In-depth look at the environment and its easy-to-use feature will be examined further in the environment theme. Since it is also related to the motivation, students mentioned that the easier the environment, the more motivated students.

Student T1_S8 emphasized that two factors affected his learning in a better way. First, simple to complex sequencing of examples, second, easygoing environment which provided students a more relaxed environment to learn. Both factors are not specific but applicable to introductory programming courses. Starting from simple examples and build it into a more complex one step by step motivates students to learn more.

> Our examples were very good. I mean, I have felt that things were like steps… We made one example in one week, the other week you reach the next level. That was very nice. For one thing, in my opinion, because the course environment was easygoing, I learned everything more easily. (T1_S8)
>
> *Örneklerimiz çok iyiydi. Yani kademelendirme olsun şeye onu bi hissettim... Bir örnek bir hafta bir şey yapıyoruz. Bir hafta bi şey yapıyoruz mesela o level atlıyosunuz. Böyle güzel oluyo o. Bir kere ders ortamı rahat olduğu için ben kendi adıma konuşayım daha kolay öğrendim her şeyi yani. (T1_S8)*

Student T1_S11 also stated that simple environment leads students to develop more willingly. Instant help and explanation of the environment also motivate students towards the course.

> Even, it looks simple a little. The simple look makes people like “I will drag this to this one, and this thing will happen” Actually, it motivates people more to develop more… When you hover the mouse, there was an explanation or the block itself was in English. And it makes you like the course because it is easy. (T1_S11)
>
> *Her ne kadar biraz basit gibi görünüyor. Basit gibi görünmesi de şey yapıyor insanda ben bunu buna getircem bu olacak. Daha teşvik ediyor aslında insanı yapmak için. Üzerinde durunca bile bloğun açıklaması ya da bloğun zaten üzerindeki yazan zaten İngilizce oluyor. yani O da dersi kolay olduğu için sevdiriyordu. (T1_S11)*

Observation notes also put forward that simple to complex building tutorial design helped students to cope with understanding new concepts and topics. Moreover, students got help from the environment through explanations of the blocks. Designing a course related to steps provided by the environment would make the course more coherent and adaptable for the students.

4.3.3 Creating a Useful/Working/Purposive Product

Strongest intrinsic motivator for the students in this course was creating a useful/working/purposeful product. Conventionally in most of the introductory programming courses, basic algorithms were taught with console applications that have no purpose other than the algorithm itself. Students who have taken other programming courses defined the products they have created at the end as meaningless, abstract, and useless. Observations revealed that from the first week to the end of the course, developing a product at the end of each week motivated students significantly towards the programming and the course. Especially, after seeing the very first working example that students created on their phone which was a button playing a sound when touched, students seemed very happy, proud and motivated. Interviews also revealed that creating a concrete product at the end is very important both for the students who have taken a programming course before and for the first-time learners.

Students, who have taken a textual programming course before, stated that they could not see the outcome of the examples. Students compared the conventional programming course with visual programming course. Student T2_S4 from Department of Business Administration, who have taken a Visual Basic course before, compared the two course regarding its products and examples. He specified that there were not any outcome of the examples they have made in that course. Even though he liked the examples, he did not see a real product at the end. This statement as the other similar ones shows that outcomes of the examples are as important as the knowledge and the process.

> I cannot see the outcome in the Visual Basic. Or, I can say that, maybe, we did not see any… We made things like Binomial distribution and stuff. Basic algorithms... Algorithms. It was also fun actually, but in this one, there is the level of creating a product. You see the outcome; it is nice in App Inventor

> because more concrete things were created. We could not make anything really, I mean there was no decent product at the end (regarding the Visual Basic course). (T2_S4)
>
> *Sonucunu ben, göremiyorum ben şeyde visual basic'te. Ya da biz göremedik belki öyle diyim. Binom açılımı falan öyle şeyler yaptık. Temel algoritmalar. Algoritmalar. O da zevkliydi aslında da bu işte ürün çıkma aşaması olduğu için. Görüyorsun sonucunu güzel oluyor App Inventor. Çünkü daha somut şeyler çıkıyor ortaya onda dediğim gibi pek bir şey yapmadığımız için yani şey olarak bir ürün çıkmadığı için ortaya doğru düzgün. (T2_S4)*

Student T2_S6 made a similar statement about his experience with C++ course. According to the students, traditional programming courses miss implementation of product creation. Students need more than algorithm examples or meaningless programs to be motivated towards the course.

> In that course (C++), we learned the basics. I mean, you learn the part like how a computer works, how you can write a program. In this one, the nice part is you have a thing to implement directly what you have just learned. In that (C++) we were writing codes like "addition program" but it is not a useful thing or things like sorting algorithms. It (C++ course) did not have a meaningful outcome. In this one (App Inventor) we get the results of what we make. A product were made and we make one or two of these every week, that was nice. (T2_S6)
>
> *O derste programlamanın artık temelini öğreniyoruz. Yani bilgisayar nasıl çalışır nasıl program nasıl yazılır tarafını öğreniyosun. Burada güzel tarafı da öğrendiklerini direkt uygulamaya koyabileceğim bir şey var. Onda işte ne yazıyorduk toplama programı yazıyorduk ama kullanışlı bir şey değil zaten. Ya da işte sıraya dizme bilmem ne. Anlamlı bir sonucu yoktu yani. Bunda direct yaptığımızın sonucunu alıyoduk. Bir ürün oluşuyordu ve de bunu her hafta bir ya da iki tane yapıyorduk o güzeldi. (T2_S6)*

On this course, both examples and the nature of the environment encouraged students to create working product instead of basic algorithms from the first example. Students with previous programming experience also criticized examples with no product. Examples or assignments with a working end-product were seen as one of the strongest motivators for the students. Students need to integrate their algorithms into their products. However, the product should not be just implementation of algorithms like sorting algorithm. End-product could contain sorting algorithm to help to solve a real world problem or just to create a game.

Students who have taken a programming course before, compared the environment they have used before with App Inventor environment. They think that the course and the environment was more theoretical in textual programming environments. Student T1_S10 states that they did not develop any application in C++ course, they have only made some algorithm examples.

> It was a little theoretical. We did not actually see… What we did appear after we print (compile) it but App Inventor is more enjoyable, we can see directly as an application. (T1_S10)
>
> *O biraz daha teorik kalıyodu aslında görmüyoduk ne yaptığımız pek aslında yazdırınca çıkıyodu ama AI tabi daha zevkli direkt uygulama olarak görebiliyoruz. (T1_S10)*

Student T1_S4 with previous programming knowledge stated that this course (or environment) was better compared to the others since it allowed them to develop a working product, rapidly.

> In other projects, we start; it took 3 to 5 weeks to put a product on the table. Or even for the smallest project, 1 to 2 days in average. (T1_S4)
>
> *Diğer projelerde şey oluyor, başlıyoruz 3 hafta sürüyor, 5 hafta sürüyor bir ürün çıkarmak. Ya da en küçük bir şey bile 1-2 gün atıyorum. (T1_S4)*

Even the students with no programming experience compared the course with other courses they have taken, and they set forth that creating something at the end of each week was a positive aspect of the course. Student T1_S6 expressed that creating something like the outcome of the course, as fun compared to the other courses. Student T2_S7 also thinks that developing a product at the end was a positive side of the course.

> I mean… The course was fun, in general, in my opinion when we compare to the others (courses)… It was focused on creating something in the laboratory. That why it is nice. It is also nice that we were the ones creating. In the end, I mean as the outcome making a product was good. (T1_S6)
>
> *Yani… Ya zaten eğlenceliydi ders bence genel olarak. Diğerlerine baktığımızda, labda tamamen bir şey yapmak üzerine, o yüzden güzel yani. Öyle bizim bir şey oluşturmamız da güzel. Sonunda yani sonuç olarak, bir ürün yapmamız güzel. (T1_S6)*
>
> When something was developed at the end, it was very nice. (T2_S7)
>
> *Sonlarında çok keyifli oluyordu, bir şey çıkınca. (T2_S7)*

Creating product is also beneficial for the assessment of the student. Project based evaluation has provided deeper assessment chance to the instructor. Students from both terms preferred project based assessment to the exams.

> Absolutely, having no exam was very good for this course. Because I have this thing, when there is an exam in a course, I am working to get good grades. When there is a project, I don't know, it is for learning. At the end, a product was emerged at least. (T1_S9)
>
> *Kesinlikle sınav olmaması bu ders için çok iyi oldu. Hocam çünkü sınav olunca şey olayı var bende de aynı şekilde sınav varsa hani bir derste sınavdan iyi not almak için çalışmak gibi bir şey oluyor. Proje olduğu zaman ne bileyim öğrenmek için oldu yani. Sonunda en azından bir ürün çıkmış oldu ortaya. (T1_S9)*

Creating something unique to herself for student T2_S3 is what motivates her towards the programming. Providing a free development environment to let students create what is on their mind is essential to motivate them towards the programming.

> What I like is creating new things, writing something unique to me, and put forward something new. I mean in my mind… Without any obligation, just transferring the thing, I put together in my mind to the computer and having an outcome from it and seeing that it works was making me happy. That was the beautiful side of programming. (T2_S3)

> *Ya benim hoşuma giden yeni bir şeyler yaratmak kendime özgü bir şey yazıp yeni bir şey ortaya koymak. Hani kendi kafam... Bir şeyin bir zorlamanın altında olmadan sadece kendi kafamda kurduğumu bilgisayara geçirip ordan bir sonuç elde etmek ve o sonucun çalıştığını görmek beni mutlu ediyordu. Buydu programlamanın güzel tarafı. (T2_S3)*

Learning that at the end of each week one or more application will be developed motivated students towards coming to class with curiousity. Since they have learned that they will create an application at the end of each tutorial, students were highly motivated. According to observation notes both students T1_S2 and T1_S9 asked enthusiastically that what they will develop this week during the course. Student T1_S11 indicated that making even a small application, in the beginning, motivated them to make something.

> Desiring to make something… We made an application in the beginning, even if it was small. (T1_S11)

> *Yapabilme isteği. Başlangıçta küçük uygulama da olsa yaptığımız için. (T1_S11)*

Knowing programming, feel competent about developing an application is also another motivator in itself for the students. It was in direct relation to creating a useful/working product. Creating a product also leads to a social motivator. It was also motivating for students to show their friends the product they have created as student T1_S10 mentioned. Moreover, student T2_S6 indicated that he showed the examples that he has made in C++ course to their friends, and they were not impressed. On the other hand, they were playing with the visual programming examples.

> I mean, now I am cool. For example, my friends come and say what are you doing, and they are like “Wow, what is that” Actually it is easy when you learn, not that easy but when other people look at it they say wow. (T1_S10)
>
> *Yani şimdi havam oldu diyomuşum diğer insanlara. Mesela arkadaşlarımdan gelen ne yapıyorsun diyo hani vay be neymiş falan diye aslında hani öğrenince basit gibi hani çok basit olmasa da diğer insanlar bakınca vay be diyo. (T1_S10)*
>
> My friends saw what I wrote in C++ and they were saying, “Is that it, you made just this” In this one (App Inventor) you show them in the phone, and they were playing it and talking about it. (T2_S6)
>
> *Ben C++ ta yazdığım programları görüyordu arkadaşlarım bunu mu yazdı. Bu mu yani diyordu. Bunda telefonda gösteriyorsun oynuyorlar bir şeyler söylüyorlar falan. (T2_S6)*

Other than creating useful applications while making the examples, some students also found it motivating to create interesting applications like games. Creating examples based on the interest of students could motivate students even more towards the course and the topic as student T2_S1 mentioned that the game examples were the most enjoyable part of the course.

> The most I enjoy because the examples are most of them are games. So, after we program it we can play it directly. It was enjoying. (T2_S1)

T2_S7 also found creating a game at the end of the class as entertaining. Student T2_S5 also thought that game related tutorials were attention grabbing.

> I love it when a game was created at the end of it, which was entertaining. (T2_S7)

> *En son bir oyun çıkınca çok eğlenceli oluyor onları çok seviyorum. (T2_S7)*
>
> Parts I liked was tutorials which were very nice. I mean, they were getting attention. I liked those… There was the one with Hulk; the other was Street Fighter. (T2_S5)
>
> *Sevdiğim yönleri şeyler tutoriallar falan çok keyifliydi. Hani dikkat çekicilerdi. Onları seviyordum. Bir yandan Hulk geliyordu Street fighter geliyordu o güzeldi hani. (T2_S5)*

Observation notes from second week of the first term showed that one of the students after finishing the example stated that "Now the button makes sense a lot, you understand after you finished." And explained the functions of each button to his friend which shows that creating product is beneficial to make abstract concepts concrete. Also it showed that it is important to show the fully working example to students before starting rather than verbal explanation. Creating useful/working/ meaningful products are essential as the interviews and observations put forward. Therefore it would be beneficial for instructors to design and develop their computer programming courses and activities for beginners by putting the product of each activity at the center as well as the learners. This approach is called product-first approach and will be investigated more in the discussion part.

4.3.4 Self-Improvement

Apart from the novelty effect, learning something new was also very motivating for the students. For both students who had no programming background and students with previous programming experience, learning new topics or features were important.

Interviews put forward that according to the first-time learners of programming, learning programming was essential. Having the sense of learning programming motivated students towards the course by itself. Student T1_S11 stated that self-improvement feeling which was from learning something she has never learned before, made her motivated more towards the course. The sense of learning something new and self-improvement was one of the primary sources of motivation for some of the students attended the course.

I enjoyed doing the activities we made. Generally, I liked them all, from the beginning to the end. I mean, the self-improvement makes you feel good… I mean, actually, since this is something I have never done, I feel like improved. That's why I liked this course. I liked it because I felt developed since it is self-improvement. I liked it even more when I learned something new. (T1_S11)

Yapılan etkinliklerden keyif alıyordum. Genel olarak yani hepsini sevmiştim ilk baştan sona kadar. Hani insanın geliştiğini hissetmesi güzel bir duygu... Yani şöyle aslında benim sevdiğim yanları daha önce hiç yapmadığım bir şey olduğu için ben kendimin geliştiğini hissettiğim için bu dersi seviyordum. Self-Improvement olduğu için geliştiğim hissediyodum o yüzden seviyordum. Bir şeyler öğrendikçe daha çok seviyordum. (T1_S11)

Student T1_S7 had no knowledge about programming, and that caused him to feel incompetent about programming. Familiarizing towards complex parts of programming improved her motivation. T1_S6 also stated that learning about programming was worthy for her since she did not have an idea what that was.

I, for example, have no knowledge of programming. It (the course) gave me that information about programming. The function of each part was too complex for me. I mean, I became familiar with that. (T1_S7)

Benim mesela programlama yönünde hiç bir bilgim yoktu. Belki programlama yönünde bana bilgi verdi. Neyin ne işe yaradığını çok kompleks geliyodu bana biraz hani ona aşina oldum. (T1_S7)

I mean at least I have an idea about programming. I didn't know anything, about what it is. (T1_S6)

Yani en azından bi fikrim olmuş oldu programlamaya dair hiçbir şey bilmiyordum nedir ne değildir. (T1_S6)

Student T1_S2 compared the homework of the course to homework of other courses. He saw the homework with a fruitful product as a source of new knowledge, which motivates him towards the course and doing homework even more.

> They weren't like classical homework. For example, what is the benefit of writing the report in community service course? But when I do this homework I am going to learn something, that is obvious, so I will do that willingly. (T1_S2)
>
> *Diğer klasik ödevler gibi değil. Mesela Community Service dersinde ben o raporu yazsam ne olur yazmasam ne olur? Ama bu ödevi yapınca bir şey öğrencem belli yani o yüzden seve seve yaparım. (T1_S2)*

For students who had some experience regarding programming, developing in a new, visual environment in which the product is for mobile devices is also another source of motivation. Having the sense of learning something new was also a strong motivator for students with previous programming experience. Student T2_S6 stated that learning new features and constant progression towards new topics keep him motivated towards the course.

> Despite that, it wasn't a course I can say like "I already know those why did I (take this), I can do this blindfolded" We constantly make something new, we put something new on the top of that. Those were fun. I mean, another thing is that ok, we know programming. We are computer education and instructional technology students, but I never feel like I knew this already, I really liked it. For example, we are working with touchscreens, its features, etc., we entered into a new environment. (T2_S6)
>
> *Ona rağmen böyle of ben bunları biliyormuşum zaten ya niye şey yaptım rahat rahat yaparım diyeceğim bir ders değil sürekli yeni bir şeyler yeni bir şey daha yaptık üstüne bir şey kattık falan onlar eğlenceliydi. Yani bizim sonuçta şu da var mesela biz işte programlama biliyoruz böte öğrencisiyiz falan ama ben derste hiç şeyi hissetmedim hani biliyorum zaten şeyi yapmadım hakkaten*

> *sevdim. Mesela şey hani dokunmatik şeylerle uğraşıyorsunuz bir taraftan onun şeyleri vs. tamamen yeni bir ortama geçmiş olduk. (T2_S6)*

During the activities, students should feel that they are learning something new. Through the new functions and features, students were kept motivated towards the course and the environment. Additionally, interviews and observations revealed that feel of self-improvement was one of the important motivators for the students. Even the students with programming experience were motivated towards developing a visual application for the mobile operating system since they were in a new environment with different dynamics and products.

4.3.5 Preferring Practice to Theory

Modern educational paradigms suggest learning meaningfully, rather than memorizing verbatim (Charles M. Reigeluth, 1999b), programming is not an exception. as the students in this study preferred the similar. Students found the tutorials meaningful and preferred the practice to theory. Student T1_S6 stated that learning functions of blocks instead of memorizing information, as raw data was a good side of the course.

> I mean it was nice. You were telling us like “this button’s function is that, and it is used for this.” so it was beyond memorizing. I mean, it wasn’t from the top of our hats, like do this and do that without knowing the function. That was nice. Your explanation of the use of the blocks and their functions. (T1_S6)

> *Yani güzeldi hani sizin de şey genel olarak şu buton böyle bu işe yarıyor bu yüzden bunu kullanılmış dediğiniz zaman hani birazcık şey olmaktan çıktı. Hani ezber olarak ordan bunu böyle yap bunu böyle yap ne işe yaradığını bilmeden. Güzeldi onlar falan. Anlatmanız blokların kullanımını… Görevini falan bu yüzden kullanılmış falan diye. Eksik olarak bilmiyorum pek bir şey yoktu. (T1_S6)*

Student T2_S1 thinks that hands-on practice from the day one of the course was very interesting. Instead of theoretical information from the start, T2_S1 stated that he prefers practice.

> The most interesting is on the first meeting when we are… has to try the games and I like the labs because we can write the codes by ourselves… And, the other is there wasn't any lecture. (T2_S1)

The researcher asked students if anything was boring regarding tutorials or any part of the course. This question also revealed the part that students liked most. As it was mentioned before under this theme, creating useful outcomes in the course is important for the students. This was also suitable for in-class activities and tutorials. Students were thinking that designing and developing an application in course hours is an advantage of the course.

> In course hours, it was good in general; I mean there wasn't anything to bore a person. Besides, in laboratory classes, all we do were designing application. (T1_S5)
>
> *Yani ders içinde genelde iyiydi yani şey olarak bi insan sıkılacağı bir nokta yoktu zaten laboratuvar dersinde tamamen şey işliyoduk uygulama tasarlıyoduk. (T1_S5)*

Students T2_S5 thinks that it was fun to create something at the end of each step in course hours.

> No, it wasn't boring It was fun. I mean at the end of each step, seeing that you made something, something was created… (T2_S5)
>
> *Yok sıkıcı değil gene eğlenceliydi hani her bir adımın sonunda bir şey yaptığını görmek bir şey oluştuğunu görmek... (T2_S5)*
>
> Creating an application in course hours was fun, not boring at all. (T2_S7)
>
> *Ders saati içinde uygulama geliştirmek güzeldi sıkıcı değildi. (T2_S7)*

According to student T1_S7, learning with hands-on practice helps students to learn the topic easily.

> When we look at it, when we try to do something, we were learning easier. (T1_S7)
>
> *Kendimiz bakarak bir şeyler yapmaya çalışarak daha kolay öğreniyoduk. (T1_S7)*

T1_S5 stated that developing applications further keeps the student motivated towards the course because of the freer environment.

> I was looking forward to making things (applications) more detailed. That part of the course was very enjoying for me. (T1_S5)
>
> *Daha detay şeyler yapmak istiyo daha ne yapabilirim gibi bakıyor. O yönden zevkli bir dersti yani benim için. (T1_S5)*

As the outcome of this code, students prefer environments and courses which provide a freer atmosphere. Additionally, practice-based learning helped students to be motivated towards the course. Starting to learn with practice and developing actual products rather than theoretical information Especially, students emphasized that learning by developing (similar to learn by doing) motivated them starting from day one. As a very similar finding to creating products at the end, students also were in favor of the creation process to learn the programming concept or the feature of the environment. Students also reported that learning through the practice made them learn the concept easily and motivated them.

4.3.6 Proofs of the Motivation/Competence

According to the observation notes, most of the students were motivated throughout the semester. Even during one course hour during the fourth week of the first semester, students choose to continue the course instead of finishing it early. Another indicator for the motivation is the will of the learners to continue to learn the subject or take it to the next step. According to the interviews, 15 out of 18 students were planning to develop their own applications for their professional life or their own. Improving their abilities or using it to develop is also another indicator of motivation related to competence. Student T1_S2 was motivated to develop his application after the course ended. He wanted to create the application he needs instead of searching the Play Store.

> If I need an application and it is a simple one, I won't search Google Play. I would try to make myself… If I am in need, as I said why not. I do not want to use the prepared one from now on. First, I want to make it. I think that I want to elaborate on it. (T1_S2)
>
> *Dediğim gibi bi uygulamaya ihtiyacım olursa çok basit bir şeyse Google Play'de aramam uğraşırım yapmaya çalışırım. Yani ihtiyaç olursa dediğim gibi neden olmasın. Direkt hazıra konmayı istemiyorum şu anda önce bi kendim yapayım. Çabalayayım tarzı düşünüyorum şu anda. (T1_S2)*

Student T1_S5 was thinking to develop his programming knowledge even further into programming in native Android programming environment.

> I am planning; I was planning to learn Java… I mean this could be a foundation for it because I have developed an application before and we can publish it in Google Store, Google Play. (T1_S5)
>
> *Ya ben şeyi düşünüyorum Java öğrenmeyi düşünüyodum… Yani onun için bi altyapı olabilir bu çünkü daha önce program yapmışım ve onu belki Google store'da da Google play'de de yayınlayabiliriz. (T1_S5)*

Regarding the course assignments, student T2_S3 stated that they were working on homework or project willingly which shows that student was motivated during the course, even when he was doing homework.

> For the project, I worked on it every day till I finish it. Last 2-3 days I sit in front of the computer like 6-7 hours because I need to debug the application. However, since I didn't get bored doing it, I wasn't affected from how much time I spent. (T2_S3)
>
> *Proje için mesela günlük bitirene kadar baktım. En son 2-3 gün 6-7 saat falan başında oturdum çünkü hatalarını flana ayıklamam*

> *gerekiyordu ama sıkılmadığım için başında ne kadar kaldığım aslında beni etkilemedi. (T2_S3)*

As one of the students, student T2_S6 summarized his opinions of the course, which were parallel with the views of other students in brief regarding the motivating part of the course.

> Creating a product, fun course hours, being outcome focused, being useful, constantly getting something new… Even if you know the programming… It was a lot of fun. Teaching method, topics, examples we make… Without any trouble or without saying "not this course again" even it was at very early hours of the morning, we came to the class. (T2_S6)
>
> *Ürün ortaya çıkarmış olmak, derslerin eğlenceli geçmesi, hani sonuca yönelik olması, kullanışlı bir şey olması, sürekli yeni bir şey hani programlamayı bilsen bile... Gayet eğlenceliydi. Derslerin işlenişi konular yaptığımız örnekler falan herhangi bir sıkıntı yaşamadan şey demeden off gene ders var yaa falan kim uğraşacak sabahın köründe falan demeden gayet geldik. (T2_S6)*

This code focused on the whether the interpretation of motivations were true, or not. An important indicator of motivation was to keep the interest on the topic. As it was mentioned the majority of the students were planning to develop application even though their affiliation with the course has ended. Some students stated that after the course ended they want to spend time on developing their own application rather than downloading a prepared one. In addition to continuing interest after the course ended, students also reported that they were motivated during the course. They wanted to come to the course as well as some of them were willingly worked on homework and projects. Motivation is an important factor for any course, especially in a topic with high drop-out rate like programming it is essential. Therefore, it is important to check the motivation level of the students during the course as well as at the end.

4.3.7 Summary of Motivation Theme

Motivating components and features will be reiterated and added based on the interviews of the students. Major motivating features of the visual programming environment for the students were being easy, having visual feedback and being able to produce visual products in the end.

- Motivating features of visual programming environment was one of the strongest motivators of the course. Which leads to
    - Easier learning
    - Make programming concepts concrete
    - Enthusiastic about learning

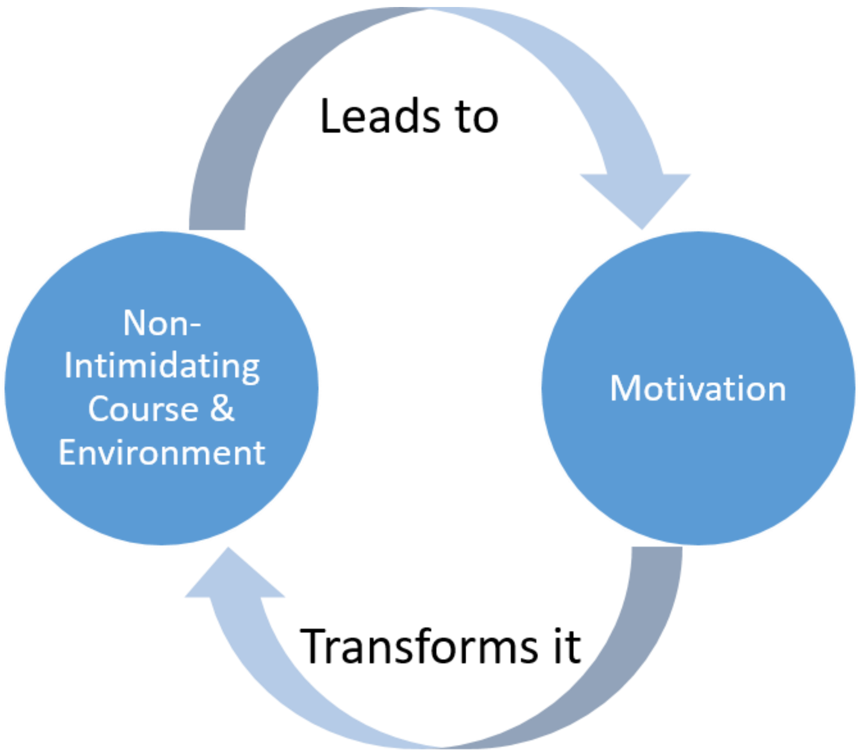


Figure 4.8 Motivation Effect of Course Environment

- Another finding emerged from the interviews and observations was designing course and tutorials according to a product-oriented design
    - Products are as important as the process and the knowledge itself
    - Students want to create useful products rather than solely algorithms
    - The proper product at the end of tutorials/homework/project will be the reward and motivator by itself
- Keeping the sense of learning new aspects and additional features of programming is necessary for the motivation of students
    - Activities in each week should be connected with the week before, but something entirely new should also be provided

- Students prefer practice to theory so
    - Hands-on practice should be provided from week 1
    - Practice should be open to discovery learning. Even the novice learners prefer to learn by practice and discovery
    - Providing a freer environment to students for developing application further than minimum expectancy motivate students while learning new information
    - Creating applications in the course hours were motivating students and help them to learn better
- Constant communication in and out of the classroom keep students motivated
    - Relaxed and comfortable atmosphere should be kept during the course hour to keep communication between students and instructor and between students
    - Outside of the classroom communication medium should be kept active to encourage the communication

4.4 Programming and Programming Environment

Since one of the most important aspects of this course was the programming environment and its features, the researcher asked questions regarding the environment and the comparison of the block-based visual programming environment (specifically App Inventor) to the other programming environments. In this way, researcher intended to reveal the difference of visual programming environments from conventional textual programming environments. In addition, since App Inventor environment was used, positive and negative sides of the App Inventor were tried to be examined based on the student perspective. This could help both developing new programming environments and enhancing existing ones.

4.4.1 Attitude Change towards Programming

Even all of the students took this course as elective, and they participated in the course voluntarily, some of them stated that they were afraid of the programming because they think it is hard to learn. Some students stated that after taking this course, they overcame their fear of programming since they are familiar with the programming concepts with the help of the visual environment. Student T1_S3 stated that for a person out of the computer field programming seems very complicated. However, this course stopped her to be afraid of programming.

> If a person is out of the area, it seems very complicated. Yes, we are more familiar with this kind of things… You know we developed an application working on the phone. (T1_S3)
>
> *İnsan böyle şeylerin dışında olduğunda insana çok karmaşık geliyor. Evet biraz daha en azında şeyiz aşinayız böyle şeylere ne bileyim telefonda çalışan bir tane uygulama yaptık. (T1_S3)*

Student T1_S2 was also expressed his attitude towards programming before and after taking the course. He stated that curiosity for the environment won over their fear. The right choice of environment could help students as much as an instructional strategy.

> We took this course because it is programming. When we see that it is App Inventor, and everything is ready to use we like it more. I mean, because we were afraid of the Programming-I course when we were taking it, but curiosity won over the fear. (T1_S2)
>
> *Şimdi bu dersi şey diye aldık programlama diye aldık AI olduğunu öğrenince her şeyin hazır olduğunu görünce daha da bir sevdik. Yani çünkü programlama 1 dönem dersine korkmuyor değildik alırken ama merak olunca korkunun önüne geçti merak aldık. (T1_S2)*

While conventional (textual) programming was seen as a complex labor by student T2_S7, visual environment made her think that programming in a visual environment is simple.

> Positive sides are, I mean my roommate is in computer engineering, what he does is very complicated for me. This is a simpler version, so it is nice. Puzzle stuff is entertaining. (T2_S7)
>
> *Olumlu yanları şey diğer işte oda arkadaşım mesela bilgisayar mühendisliğinde onun yaptıkları çok karmaşık geliyor bana. Bu*

> *böyle daha basit hali. İyi oluyor o yüzden. Puzzle falan eğlenceli. (T2_S7)*

Observation notes also showed that most of the students started the course with lack of self-confidence regarding programming. After 2 weeks, especially after they saw that they could develop functioning applications such as games, their attitude and confidence changed towards a positive state.

4.4.2 Textual vs. Visual (App Inventor) Programming

Students, who have taken a textual programming course before, were asked to choose one of the programming environments: visual or textual. In a total of two semesters, ten students have had programming background before taking this course. Among those students, four of them preferred App Inventor, the remaining six answered the question diversely. Two of them preferred textual environment (C++ for them), other two preferred textual, however, for easier tasks they stated that they would use App Inventor, and the remaining two students preferred a combination of both environments. The reason behind their preference will be examined further under this sub-theme. Opinions of the students with no programming background were also taken if they would like to put forward their ideas. They will be mentioned in the codes under this sub-theme.

4.4.2.1 Visual Only

While some of the students thought that after taking this course, they overcame their fear and they can continue to learn programming in a textual programming environment, others find App Inventor was enough for them and still think conventional programming was too difficult for them. Student S1_T1 thought that App Inventor is sufficient for him and C++ course would be difficult for him. However, he also thinks that he can take an introductory C++ course from scratch just to discover it. He explained his negative attitude because to whomever he asked about programming, they answered ‘it is very hard.’

> AI is enough in my opinion. I mean, I think C++ will be too hard… I can take an introductory course starting from scratch to learn a little, but C++ sounds scary. (T1_S1)

> *AI yeterli bence C++ hani çok zor olur diye düşünüyorum… Sıfırdan başlayan bir tanıtım dersini alabilirim biraz öğrenmek için bir şeyler ama C++ deyince bi gözümüz korkuyor. (T1_S1)*

Syntax Related Problems

Some students preferred visual programming to textual because they think that syntax is a waste of time, and constantly causing (punctuation errors, etc.) problems. T1_S8 who have taken and failed a programming course once stated that developer must write the code without a single character mistake to get the software to work. This was seen as a negative side of textual programming environments.

> You must write everything up to the smallest detail (in textual programming environment). OK, there are auto-complete things in some software, but still, you need to write. (T1_S8)

> *Her şeyi çok en temel şeyine kadar yazmak zorundasınız yani. Tamam, tamamlama olayları oluyor bazı şeylerde, programlarda ama yine de yazmak zorunda kalıyorsunuz yani. (T1_S8)*

Student T2_S3 who is also a CEIT student with programming experience expressed his choice toward the visual programming environment because of the syntax errors in textual programming environments.

> I would have chosen the visual one of course... Visual Blocks because writing syntax is a waste of time. Because making the slightest mistake makes it difficult to return and look, if the program is too long. (T2_S3)

> *Tabi ki görsel olanı tercih etmek isterdim… Görsel bloklar, çünkü vakit kaybı syntax olarak yazma. Çünkü en ufak bir hatada belki geriye dönüp bakmak program uzunsa zor olabiliyor. (T2_S3)*

Immediate Feedback and User-Friendly Environment

Student T2_S3 thought that immediate feedback and instant correction makes the programming easier and time-saving.

> However in here (App Inventor) you get immediate feedback, you see that there is a mistake and you have the chance to

> correct that instantly. While connecting the blocks compatible ones connect, but for example, if they are wrong they do not connect. That gives a hint to the user. Being visual is not time-consuming, it is time-saving and makes it even easier. (T2_S3)
>
> *Ama burada anında geri ileti alıyorsunuz hata olduğunu görüyorsunuz anında düzeltme şansınız var. Blokları birleştirirken aslında birleşebilecek olanlar birleşiyor ama mesela yanlışsa birleşmiyor o da bir ipucu veriyor karşıdaki insana. Görsel olması hem vakit kaybı değil kazanç sağlıyor hem de daha kolay hale getiriyor. (T2_S3)*

Being easy, visual and having some kind of scaffolding system through blocks, which prevents users from making simple mistakes, another reason for choosing App Inventor environment over textual environments like C++. Student T2_S5, who have failed from C++ course once, thinks that she feels more in control in a visual programming environment. She explains her experience with both environment as follows.

> C++ was giving a hard time. I mean I didn’t know anything about it and I have a task at hand. But, we don’t have any steps. I need to think unbelievably, a lot. I did many mistakes. However, when I think Android (meaning App Inventor) they are like puzzle pieces so I can get rid off some of the mistakes at least when I was placing them. (T2_S5)
>
> *C++ bayağı zorluyordu. Hani hiç bilmediğim için direkt elimde bir görev var ve adımları yok ki bu, inanılmaz derecede düşünmek zorunda kalıyordum. Çok fazla yanlış yaptığım da oluyordu. Ama Android’I (App Inventor) düşündüğümde puzzle parçaları gibi olduğu için yanlışları birazcık daha atlatabiliyordum en azından yerine yerleştirmede. (T2_S5)*

Concrete Products

Student T2_S4 thinks that App Inventor is a simplified version of the programming, other than being unstable he did not believe that it has any negative side. He explains his preference as App Inventor allows him to produce products that are more concrete while textual environment did not allow him to create any decent product. As it was mentioned in the motivation part, one of the most important motivators for the student is to create a usable product in the end. It also affects their choice of environment for their future.

> Q: If you had the chance of choosing only one of them, which one would you prefer?
> T2_S4: I would prefer App Inventor.
> Q: Because of what?
> T2_S4: Because more concrete things are created in that one. As I said before (regarding the textual environment), we didn't produce any decent product in that one… Because it is a simplified version. There isn't any negative side for me. Just, It seems that App Inventor is not that stable yet. (T2_S4)
> *Q: Peki ikisi arasında bir seçim yapman gerekse hangisini seçerdin*
> *T2_S4: App Inventor'ı seçerdim ya.*
> *Q: Ne için?*
> *T2_S4: Çünkü daha somut şeyler çıkıyor ortaya onda dediğim gibi pek bir şey yapmadığımız için yani şey olarak bir ürün çıkmadığı için ortaya doğru düzgün… Basitleştirilmiş hali olduğu için. Olumsuz yanı yok benim için. Şey AI'ın sadece o biraz stabil değil sanki daha. (T2_S4)*

Student T1_S4 thinks that choice of the environments for him depends on your professions. He indicated that if you are not a professional programmer, App Inventor is better to create basic applications; if your career choice is programming, then textual programming is better.

> If your profession is programming, I really prefer traditional method (syntax-based programming), writing code, but if I am a teacher or my purpose is creating little applications for daily use, I will prefer App Inventor. (T1_S4)
>
> *Eğer işiniz şeyse programcılıksa geleneksel yöntemlerle gerçekten kod yazarak onu tercih ederim ama öğretmensem veya hani programlamayla ilgili sadece hani derste kullanıcam ya da gündelik*

hayatta kendimin kullanacağı küçük uygulamalar yapmaksa App Inventor'ı tercih ederim. (T1_S4)

4.4.2.2 Combination of Both

Student T2_S1 thinks that both types have the advantage and disadvantage of their own. He thinks that it would be better if a combination of App Inventor and C++ would be provided. Even if it is impossible to create an environment from scratch for an instructor, a course that teaches both simultaneously could be created.

> Actually, AI is good but the commands are still basic, so it is hard for me. I think if we can combine both. Some benefits of C++, and Some benefits of AI… I like to find the other program for Android application inventor. If there is one… (T2_S1)

Student T2_S5 stated that it would be better to have a combination of both environments. To be exact, she thinks that App Inventor would be better if it allows the user to write code when needed.

> T2_S5: I would have mixed both. If there was an area to write code, if needed, it would be much better in my opinion.
> Q: You mean like App Inventor interface, but you can check (the code)?
> T2_S5: Exactly, I mean like having puzzle pieces, but on the other side there could be codes. (regarding tabs) it could have three steps. However, App Inventor outweighs (C++) frankly.
> *T2_S5: İkisini karıştırırdım .evet gerektiği yerde kod yazabileceğim bir alan da olsa çok güzel olurdu diye düşünüyorum*
> *Q: Yani AI arayüzü ama açıp da bir bak gibi*
> *T2_S5: Aynen yani şey gibi bir yandan puzzle parçaları gibi olsun bir yandan kod olsun bir yandan da dizayn yani 3 aşamalı olsun gibi. Ama AI daha ağır basıyor açıkçası*

T2_S2, a student with advanced programming knowledge, stated that App Inventor has its benefits like efficiency. However, he also thinks that it is not enough for advanced development. If App Inventor will reach to the complexity of conventional programming environments, he says that he could choose the App Inventor over textual programming environment. He states that App Inventor has a smooth learning curve; however, it has some limitations for advanced programming.

> …The difference from 'normal programming' is that preparation of one could take nearly six months, you will complete preparation for App Inventor in just 1-2 weeks because it is more visual type in general. I mean that in terms of time and getting used to, App Inventor provides a very big advantage. In addition, as a user, you can proceed up to some point in App Inventor, but you stuck at some point. It needs more advanced stuff. That is a thing related to software's self-development. If the software can develop itself, I mean if it can equalize itself with regular learning strategies, learning with App Inventor will be much better. (T2_S2)
>
> *Normal programlamayla olan farkı birisindeki ön hazırlığın atıyorum neredeyse 6 ay kadar sürmesi, yani sürebilecekken App Inventor da sadece genelde görsele dönük olduğu için 1-2 haftada o ön hazırlığı tamamlamış oluyorsun. Yani zaman ve alışma açısından çok büyük bir avantaj sağlıyor App Inventor. Kullanan olarak da belli bir yere kadar gelinebiliyor App Inventor da ama bi yerde tıkanıyor daha gelişmiş şeyler istiyor. O da işte programın kendini geliştirmesi ile alakalı bir durum. Program kendini geliştirirse yani birebir endeksleyebilirse, normal öğrenme yöntemleriyle App Inventor'la öğrenme yöntemi çok daha iyi olur. (T2_S2)*

Student T2_S6 reported that while C++ course is more of a theoretical side of computers and programming, it does not provide anything useful for long term. The better side of the App Inventor course was providing them a chance to implement what they have learned into an application.

> In that course, we can learn the basics of programming, I mean, you learn how computers work how codes are written part. In this course, better side is that there is a place that you can implement what you have learned directly. In that one (C++) we write applications that sum the numbers, but it is not a

> useful thing. Or, like sorting algorithm, we make it but it does not have anything in the long term. (T2_S6)
>
> *O derste programlamanın artık temelini öğreniyoruz yani bilgisayar nasıl çalışır nasıl program nasıl yazılır tarafını öğreniyosun. Burda güzel tarafi da öğrendiklerini direct uygulamaya koyabileceğim bir şey var. Onda işte ne yazıyorduk toplama programı yazıyorduk ama kullanışlı bir şey değil zaten ya da işte sıraya dizme bilmem ne hani şeyi yok yapıyoruz ama bize uzun süreli bir şeyi yok. (T2_S6)*

Using a textual programming environment with a visual one is another strategy to make students familiar with textual environments. Using both visual and syntax-based programming languages could be beneficial. It could help them to see the similarities and differences. Also, it could assist them to use App Inventor as a step for the syntax-based programming language.

> I mean that is written in that software like this. In App Inventor, you can make the same thing as this. Similarities can be shown like that. Or, after showing it (syntax), it could help as a guide like how can it be done in App Inventor. (T1_S11)
>
> *Hani o programda bu böyle yazılıyo. Bizim app inventorda bu böyle yapılıyo arasındaki benzerliği gösterip ya da önce onu gösterdikten sonra acaba app inventorda nasıl yapıyoruz diye bi yol gösterici oalrak ona yardım edebilir. (T1_S11)*

4.4.2.3 Textual Only

More Flexible and Advanced

As it Student T1_S5 because of the limitations of App Inventor, it is better to choose a textual programming environment like C.

> I mean I would prefer C probably; you can create more comprehensive things. App Inventor is a more limited software; you cannot develop anything you want. As I said in the presentation, you can't even change the font type to the type you want. (T1_S5)

> *Yani ben C'yi tercih ederdim herhalde yani daha kapsamlı daha fazla şey yapabilirsiniz. AI biraz daha kısıtlı bir program gibi her istediğinizi yapamıyorsunuz. Geçen şeyde de sunumda söylediğim gibi yazı fontunu bile tam istediğiniz şekle getiremiyosunuz. (T1_S5)*

Student T1_S8 also prefers C language because of its flexibility. However, he also stated that App Inventor is very useful to develop a simple application. He thinks that it is better to be specialized in a textual programming environment instead of a visual one for his career.

> I would prefer C because of the flexibility. And, by flexibility, I mean, I have an application in my mind, and I can develop it ideally by writing the code. However, in App Inventor, it is very good regarding time. Let's say, I have a simple application in my mind, I will use App Inventor to develop that. I can do that in no time. (T1_S8)

> *C yi esneklikten ve esneklikten kastım şu: Mesela benim kafamda bi program var ben bu programı en iyi yazarak yapabilirim şu anda. Ama AI da da zaman olarak çok iyi mesela basit bi program var kafamda bunu yapmak için AI kullanırım. Çok kısa sürede yaparım. (T1_S8)*

Student T1_S3 with no programming background before this course thinks that textual programming environment could be more flexible.

> It looks like we can transfer what we have in mind in there (textual programming), but in here we are limited to blocks so that it could be a more narrow area. (T1_S3)

> *Sanki orası daha böyle şey gibi aklımızdakini daha istediğimiz gibi aktarabilirmişiz gibi geliyor ama burda bloklarla sınırlı olduğumuz için daha dar bir alan olabiliyor hani. (T1_S3)*

Even though student T2_S2 was thinking that visual programming environment has many advantages like being easy and faster development, he stated that he would prefer textual because App Inventor would not let him advance as the textual programming environments do.

> If I look at the current condition, I will choose the traditional one, because I have learned what it has got to teach me already. Because I couldn't go further, I would be in the position of choosing a normal/traditional one... It is a fast solution, and it provides easiness about many things. However, if it (application) was more advanced until App Inventor improves itself, I cannot make anything. (T2_S2)
>
> *Şu anki durumuna bakarsa gelenekseli seçerdim çünkü ben AI ın bana öğreteceğini zaten öğrenmiş durumda olduğum için orada daha fazla yere gidemediğim için normal gelenekseli tercih etmek durumunda kalıyorum… Hızlı çözüm hem de kolaylık saladığı için çoğu şeyde. Ama eğer ki kapsamlıysa AI kendini geliştirene kadar bir şey yapamam. (T2_S2)*

Student T2_S6 also stated that textual programming is more flexible and provides a wider area to go further than the visual programming.

> You can make something very complex, and complicated in C++ or another programming (environment). It can be used in a broader scope. It has a bigger elbowroom. It is not like the blocks in this one, in the end, you can do anything you want. (T2_S6)
>
> *C++ ta ya da diğer programlamalarda çok kompleks böyle karmaşık bir şey yapılabilir. Daha geçiş çaplı kullanılabilir. Hareket alanı çok geniş. Bundaki bloklar gibi değil sonuçta her istediğini yapıyorsun. (T2_S6)*

As the summary of textual vs. visual sub-theme, some important points will be revisited. Students emphasized the advantages of using a purely visual environment. First of all, textual environments found harder to use and understand for students, and writing syntax is found time-consuming. Moreover, visual environments give immediate feedback and in environments like App Inventor applications could be tested without compiling. Similarly, students think that having a user-friendly design helped them to reduce their mistakes. One of the most important advantages, as it was mentioned under the motivation theme, is the ability to develop a concrete product at the end.

The only advantage of using solely textual programming comparing to the visual environment was to have a more flexible and open environment for advanced programming. Students reported that they are limited to App Inventor’s features when they compare it to a textual environment like C++.

Some of the students agreed with the advantages above, however, they also reported that combining both visual and textual environments would be more beneficial. An environment which is easy to develop a visual product at the end and also advance enough to develop a more complex application. Some students just needed to have another tab to edit the code textually while keeping all other features. Another idea was to combine the learning with both environment. It could be helpful for students to learn some of the programming concepts textual and implement it in a visual programming environment.

4.4.3 Evaluation of the App Inventor Environment

While the comparison of the textual and visual environment was made in the previous sub-theme, under this section includes the opinions of students about the App Inventor environment specifically. Evaluation of the environment from the perspective of the students could provide insight for the instructors who are going to give a lecture or create a course to choose an environment, as well as for the developers who want to enhance or who will create a visual programming environment. Students gave their opinion about the environment that they have used in this course, which is MIT App Inventor in this case. They evaluate the environment regarding its positive and negative sides. Among the opinions of students, three positive and three negative aspects of the environment come forward.

4.4.3.1 Positive Aspects

App Inventor is Visual / Simple / Productive

Three mostly mentioned positive features of App Inventor were directly in relation with each other. Therefore, they were examined under the same code. Visual nature of App Inventor was found easy to develop programs by the student. Easy to develop feature leads to a more productive environment comparing to the other environments.

As it would be broken down to investigate further, the mostly mentioned positive feature of App Inventor environment was being easy. By saying ‘easy’ students mentioned easy-to-learn, easy-to-use, and easy-to-develop. App Inventor is found easy to use, learn and develop environment even for the novice programmers.

Students with no programming experience found App Inventor easy as the student T1_S1 who was an Elementary Mathematics Education student. He stated that they use the environment to produce applications without hassle.

> App Inventor environment was simple… We use App Inventor. It is not a thing that will tire to use. Something to think a little and produce… It was easy. It was really, easy to do something. (T1_S1)
>
> *AI ortamı şeydi basitti. App Inventor’a giriyoruz çok bizi yoracak bir şey değil biraz düşünüp bir şeyler üretmek. Kolaydı. Gerçekten kolaydı onları yapmak. (T1_S1)*

Easy-to-develop nature of App inventor made students think that it is also suitable for children. Student T1_S2 thought that even children could develop an application in App Inventor environment.

> Even children could make something. With a little thinking, contemplation… I mean, they may have some difficulties in blocks part, most of the people could make something in the designer part. (T1_S2)
>
> *Çocuklar bile yapabilirdi bir şeyler. Azıcık düşünse kafa yorsa. Ha bloklarda belki sıkıntı çeker ama designer kısmında çoğu insan bir şeyler yapabilirdi orda. (T1_S2)*

Being easy and visual were related based on the opinions of the students. Since the development interface is visual, it provides some benefits for learners. Student T2_S4 stated that visual coding with puzzle pieces simplified the coding part, which shows that visual programming structure of App Inventor makes the programming simpler.

> I mean codes being puzzle pieces was nice. I mean it is a simplified version. (T2_S4)

> *Yani şey olarak kodlar olarak hani puzzle şeklinde olması bence güzel AI ın yani kolaylaştırılmış hali. (T2_S4)*

Student T1_S8 stated that visual interface prevents syntax errors as well as it has a smooth learning curve comparing to the textual environments which save the time of learners.

> In App Inventor, you don’t use the keyboard. It is a very good thing actually. It is faster, it prevents simple punctuation (syntax) errors and it is better to understand… You can make the same thing in 15 minutes, which took 1 hour on the other side (traditional programming). It shortens the time a lot. (T1_S8)

> *AI klavye kullanmıyosunuz çok güzel bir şey aslında bu daha hızlı ve işte şey basit noktalama hatalarını engelliyor ve anlaşılma şeyi daha güzel... Diğer tarafta 1 saatte yapacağınızı burda 15 dakkada yapabilirisiniz mesela. Çok kısaltıyor zamanı. (T1_S8)*

Visual programming was one of the most mentioned features of App Inventor. While students who have programming background found the code screen or command console of the textual environment to test the application boring, visual environment and outcome of the App Inventor was one of the expected positive sides of the environment. Instead of writing the code, dragging and dropping the blocks was better according to the students. It was both seen as simple and fun. Student T1_S5 who have programming experience before this course found the visual environment more interesting compared to the textual one. In addition to the development screens, he also stated that the outcome of the textual environments was worse since they were viewed in a black and white console screen.

> For example, drag and drop is better than writing the code. Because there is the drag and drop in App Inventor and it is visual, it is more interesting. In the other, you just write code. Codes are working on a black screen. This was better than that. (T1_S5)

> *Mesela sürükle bırak şeyi daha iyi bence sade kod yazmaktansa o daha iyi. App Inventor'da sürükle bırak olduğu için görsel olması da ilgi çekici oluyor. Diğerinde sadece kod yazıyosunuz. Siyah bi ekranda kodlar çalışıyor. Bu daha iyiydi ona göre. (T1_S5)*

No-syntax/only visual structure of App Inventor made student T1_S4 to think that App Inventor was different from conventional programming environments. Coding with dragging and dropping the blocks was seen as an easy thing to do.

> App Inventor is a really, different environment in my opinion. Until now (this course) you have to write the code if you want to use an environment. But in this, drag and drop… (T1_S4)

> *App Inventor zaten çok bence farklı bir ortam. Şu ana kadar hep şeydi illa ki kod yazacaksın hani. Bir ortam kullanacaksan ama burada sürükle bırakla… (T1_S4)*

Student T1_S3 also found drag and drop structure of App Inventor as very helpful especially for first-time learners.

> App Inventor is very easy to use, in my opinion. I mean, it does not have any complexity after working on it a little… For us, I mean who do not have any experience, using programming, understanding it is more complicated, more difficult. Therefore, it is more suitable for us because it uses blocks. Because it has a drag and drop method… I mean it lets you develop an application without learning any programming language. (T1_S3)

> *Zaten şey hani AI çok kullanılması kolay bence. Hani karmaşıklığı yok yani üzerine biraz çalıştıktan sonra… Bizim için mesela hiç almayanlar için bir programlama kullanmak çok daha karmaşık daha zor yani onu anlamak. O yüzden bloklarla olduğu için bizim için daha kullanışlı o var sonra sürükle bırak yöntemiyle olduğu için. O şekilde. Yani programlama dili öğrenmeden bi uygulama geliştirmeyi sağlıyor. (T1_S3)*

Other than being easier to develop, students also think that App Inventor is easier to learn. Student T2_S2 who is a Computer Education and Instructional Technology (CEIT) senior specifies the positive sides of the App Inventor environment as being easy to learn and easy to use. He said that programming turned into something for people out of the computer field.

> As my opinion, I just realized programming was transformed into something easy-to-learn for normal people. (T2_S2)
>
> *Görüş olarak sadece yani yazılımın diğer işte normal kişiler için daha kolay öğrenilebilir dönüştüğünü işte farkettim sadece. (T2_S2)*

Another CEIT senior (T2_S3) thought that App Inventor was understandable and anybody could learn it. He thought that App inventor was structured in a way that anyone could understand. Student T1_S6, on the other hand, stated that App Inventor speaks our language. While textual programming languages were seen as foreign languages to students, App Inventor uses human language, instead of computer language.

> Being visual, being in a comprehensible language… Simple at least, being in a structure that everyone can understand. (T2_S3)
>
> *Evet görsel olması anlaşılır bir dilde olması. Basit en azından herkesin anlayabileceği bir yapıda olması. (T2_S3)*
>
> It speaks our language, I mean like “we did this, when I drag this, this will happen” Because of that, it makes our job easier. (T1_S6)
>
> *Bizim dilimizden konuşuyor yani şimdi böyle yaptık sürüklendiğinde şu olsun bu olsun gibi şeyler o yüzden biraz daha kolay oluyor işimiz. (T1_S6)*

Student T1_S8 who was a CEIT student exemplified easier learning of App Inventor with ‘If’ concept in programming, in addition to the no-syntax structure of it with no need to use the keyboard. Turning the code of ‘If’ concept to visual blocks which have to be nested each other visually for multiple use of ‘If’s could help the students to learn more easily.

> In App Inventor, you do not use the keyboard. This is a very good thing actually. It is faster and prevents syntax errors. Understanding it is better, for example, blocks nest in each other. If, for example; “if” is a more understandable thing in App Inventor. (T1_S8)
>
> *AI klavye kullanmıyosunuz çok güzel bir şey aslında bu daha hızlı ve işte şey basit noktalama hatalarını engelliyor ve anlaşılma şeyi daha güzel. Mesela içiçe geçiyo o bloklar falan işte bu bunun içinde işte if mesela if AI da daha iyi anlaşılabilir bişey aslında. (T1_S8)*

Easy-to-develop feature was also found helpful by student T1_S4 who was a senior in CEIT department. Especially, comparing to the textual programming environments, he finds App Inventor more practical and rapid way to develop an application. As it was mentioned in comparison with textual environments part, App Inventor also prevents simple punctuation errors made by the novice programmers generally, due to its visual nature.

> First, in traditional programming languages, you have to spare some time, like 3-5 days. However, in App Inventor, so that there are a direct explanation and stuff in the codes, you do not have to know a lot… Of course, doing this in traditional programming languages took a little more time, wants a little more labor. (T1_S4)
>
> *Öncelikle, bu geleneksel programlama dillerini birazcık vakit ayırmanız lazım, bir 3-5 gün. Ama AI da. Direkt zaten açıklamaları filan olduğu için kodlarında birazcık da çok bilmenize gerek yok… Tabi bunları diğer geleneksel programlama dillerinde yapmak birazcık daha zaman alıyor birazcık daha emek istiyor bence. (T1_S4)*

As the programming environment, App Inventor was seen as a more productive development environment. Comparing to the textual environment he used in another programming course; student T2_S4 found App Inventor environment being more productive as one of the positive features of it.

> This course was more pleasant in my opinion because in another one we did not produce anything really, just simple things. (T2_S4)
>
> *Bu ders daha zevkliydi bnce çünkü onda hani pek bir şey hani çıkarmıyorduk. Hani basit şeyler yapıyorduk. (T2_S4)*

In addition to being visual, interface and arrangement of the environment was also found helpful by some of the students. Student T1_S2 thinks that separating the block and the designer part, and splitting the blocks and components into categories helped him to find the block or component he needed easily.

> Being visual, I mean, the interface was good. Blocks, designers; they were separated. I know what to do in where, where to add something. Categories and stuff are on the left. It was very well-arranged. (T1_S2)
>
> *Görselliği yani, şeyi iyiydi arayüz iyiydi bloklar olsun designer olsun ikiye ayırmışlar. Nerde ne yapmam gerektiğimi biliyordum. Nerden ne ekleyeceğim. Kategoriler falan solda. Kesinlikle çok iyi düzenlenmişti. (T1_S2)*

Two more students, T2_S3 and T2_S5, also put the same opinion forward as a positive feature of the App Inventor environment. Separating code (block) and design environment was an advantage according to the students.

> Design and Blocks screen tabs being separate. (T2_S3)
>
> *Hem görüntü hem blok ayrı ayrı olması ekranların. (T2_S3)*
>
> Design and Block tabs were nice. (T2_S5)
>
> *Design ve Block olması güzeldi. (T2_S5)*

Instant Feedback and Testing

App Inventor uses Blockly library. Blockly is a client-side JavaScript library for creating visual block programming languages and editors. Instead of textual codes, using puzzle-like pieces that prevents students from making basic mistakes since incompatible puzzle pieces will not interlock. This feature provides free scaffolding for novice programmers who could easily make mistakes. Student T1_S9 stated that preventing users to snap incompatible blocks corrects student's action into the relevant way.

> Blocks do not snap when they are not compatible with each other. We can say that 'Oh. This is not happening because they are not compatible' (T1_S9)
>
> *Birbirine uymayan bloklar yan yana getirdiğimiz zaman birleşmiyor. 'Ha bunlar olmuyor uyumlu değilmiş' diyebiliyoruz. (T1_S9)*

Student T2_S5 put forward a similar opinion. Preventing students from joining non-compatible blocks helped them to understand and overcome their mistakes.

> But, when I think of android (App Inventor) I can get over the mistakes a little more because they are like puzzle pieces. At least when I was placing them. I mean, you can see which part places at where. Sometimes it won't fit. You say "Ouch I made a mistake" and you can take it back. (T2_S5)
>
> *Ama androidi düşündüğümde puzzle parçaları gibi odluğu için yanlışları birazcık daha atlatabiliyordum en azından yerine yerleştirmede. Hani neyin nereye yerleşeceğini görebiliyorsun bazen yerleşmiyor. Ovv yanlış yaptım deyip geri çekebiliyorsun. (T2_S5)*

Students also think that information, which was given by the App Inventor about the blocks, was very beneficial for them. Student T1_S9 found instant feedback provided by the environment helpful.

> We can see it directly. Guidance is very good. It gives detailed information like do this if this does not work. If you say that you want to use it, I don't think you will have any problems regarding guidance. (T1_S9)

> *Direkt görebiliyoruz. Yönlendirmeleri falan filan da çok iyi bu olmazsa 'bunu yap, bu olmazsa bunu yap' şeklinde ayrıntılı bir şekilde bilgi verilmiş. Yani ben bunu kullanmak istiyorum diyorsanız hocam, sıkıntı yaşayacağınızı sanmıyorum yönlendirme açsından. (T1_S9)*

Description of the blocks and components when hovering the mouse pointer over them and information about the errors helped students to learn from their mistakes and prevents them from making common mistakes. Student T1_S11 found the description balloons beneficial for correcting errors of the learners.

> When you hover over the block, what it does is written. Let's say you made one (mistake). At least you can instantly see your mistake. You could think about what you should do next, I mean you can change it to something else. (T1_S11)

> *Bloğun üzerine geldiğiniz zaman ne yapacağı yazıyo. Hani bir sefer yaptınız. En azından yanlışınızı hemen görebiliyorsunuz. Sonrasında ne yapmanız gerektiğini düşünüp hani onu çevirebilirsiniz başka bir şekilde. (T1_S11)*

Even the students with a programming background found the supporting of the environment to prevent users from making mistakes as very beneficial. Through compatibility of the puzzle pieces and warnings and errors at the bottom of the blocks screen, users can overcome their mistakes by themselves without the further guidance of the instructor. Student T1_S4 stated that he evaluated and corrected his own mistakes with the help of feedback and support from the environment.

> Because of that, I think that this does not belong to here, it is easy because of that. And also, on the left downside it shows warning and errors, so it became more, it became easier. (T1_S4)

> *O yüzden demek ki buranın değilmiş bu hani diyorum o yüzden kolay. Bi de hani bu şeyler sol alt köşede warningler errorlarda gösteriyordu o yüzden çok daha şey oluyordu kolay oluyordu. (T1_S4)*

Another positive aspect of the environment according to the students was instantaneous testing through the phone. Application in App Inventor does not have to be compiled. With the help of App Inventor companion app, users can instantly see the change they have made on the phone. Student T1_S4 stated that trying and playing with the application they have developed at the end of the course instantly was a positive aspect of the environment.

> We can instantly try them. We can see the applications instantly on our phones at the end of the course. We can even play with them. (T1_S4)
>
> *Hemen deneyebiliyoruz. Uygulamalar hemen son günde de, ders sonunda da telefonumuza yükleyip görebiliyorduk oynayabiliyorduk hatta. (T1_S4)*

Student T1_S5 also stated that testing the application instantly and simultaneously in phone screen while developing it was another positive feature of the environment.

> Seeing directly on the phone is a nice thing… But I can see the changes I've made directly from the phone. I mean you made something and instantly see the result. (T1_S5)
>
> *Doğru telefonda direkt görmeniz o da iyi bir şey Ama o yönden iyi yani direkt telefondan görebiliyorum yaptığım değişiklikleri. yani yapıyosunuz hemen sonucunu görüyosunuz. (T1_S5)*

Student T2_S3 and T2_S1, both CEIT students, found testing the application and seeing the outcome while creating it without any need of compiling as it was in conventional programming environments as an interesting and better side of the App Inventor environment.

> As I said, when I write the code in screen and start it in an android tablet or phone seeing the output of what I write instantly, was beautiful regarding programming, in my opinion. That's what got my attention the most. (T2_S3)
>
> *Dediğim gibi o kodu yazıp ekranda çalıştırdığım zaman mesela andrroid bir tablette ya da telefonda çalıştırdığım zmaan onu*

yazdığım şeyin dönütünü görüyor olmak anında o çok güzel bir şey bence programlama için. En çok dikkatimi çeken buydu. (T2_S3)

AI better because we can play it directly from the phone and try it if it's working… There is no compiling. So it's better in AI. (T2_S1)

Summary of positive aspects

Positive sides of the App Inventor environment will not reveal the advantages of the environment per se. It also holds the potential for the relevant programming environment choice for the future courses, especially for the introductory programming courses. Students reported that the simple structure was one of the remarkable positive aspects of the environment. Especially for the new learners, easy to understand and develop aspect of the App Inventor environment was crucial for the students. Moreover, using visual puzzle pieces to develop the application was found advantageous which prevents users from making syntax mistakes. According to students, developing visually and constructing a visual product at the end helped students to learn faster.

Another positive aspect was instant feedback and testing. Descriptions provided by the environment for each puzzle piece and component. The puzzle pieces were also found useful regarding providing feedback to the students, since some of the puzzle pieces compatible with each other while others are not. This provided scaffolding to students when they were confused about using which puzzle piece in where. It can be seen that from the statements of the students, features of the environment is essential for an introductory programming course, especially in terms of being simple, helpful, communicative and productive.

4.4.3.2 Negative Aspects

Opinions of the students about the negative sides of the environment were also taken through the interviews. Students reported the negative aspects of the App Inventor environment as not being flexible enough for advanced programming, having technical deficiencies and lacking visual flexibility. Those aspects will be examined with the quotes from the interviews of the students.

Not Flexible Enough for Advanced Programming

As it was mentioned in the comparison of textual and visual programming environments, the most common answer from the students with previous programming experience is that App Inventor is not flexible enough for advanced programming. Even though they find App Inventor useful for introductory and practical level; they do not think that it is relevant for further improvement and professional career. Student T1_S5 thought that visual programming would not offer enough opportunity as textual programming does.

> App inventor seems to have no significant negative side. Just maybe, we could not make everything we could make with C++. (T1_S5)
>
> *AI da yani çok bi olumsuzluk yok gibi sade belki C++ta yaptığımız herşeyi bunda yapamıyor* olabiliriz. *(T1_S5)*

Student T1_S8, on the other hand, was thinking that creating the codes with the visual interface will not be free and flexible as it is in textual programming environments.

> You still do not feel like as flexible as writing with the keyboard. (T1_S8)
>
> *Hala şey kadar esnek hissetmiyosunuz yazarken klavyedeki kadar esnek hissetmiyosunuz. (T1_S8)*

Some students think that reason behind being not flexible enough for advanced programming was being online. They believe that if it became an offline application, the problem will be solved as students T1_S4 and T1_S5 stated. Therefore, another downside for them is being solely online.

> Of course, App Inventor has negative sides. Because of working online, we cannot make very complex things in bigger projects. It limits us. (T1_S4)
>
> *Şimdi şeyin AI ın olumsuz yanları tabi online olarak çalıştığımız için daha büyük projelerde onla zaten çok bir şeyler kompleks işler yapamıyoz bizi kısıtlıyor. (T1_S4)*

> App Inventor could be better as an application on the computer, not internet-based. (T1_S5)
>
> *App Inventor'ı uygulama şeklinde yani bilgisayarda uygulama şeklinde olması da daha iyi olur bence internet üzerinden değil de. (T1_S5)*

Technical Deficiencies

Students also think that App Inventor has some technical deficiencies. Especially shortcomings that prevent users from making a change he or she wants, crashes and errors, lack of some basic features were listed as technical problems of the App Inventor environment by the students. Student T2_S2 states that lack of some features restrict them to make the changes they want.

> For example, there are restrictions… However, in traditional programming, you can make anything you want. You can make things like open the first window, close the second and open the fifth but App Inventor does not have this feature and restrict you, so constantly stack of screens pile up. App Inventor has shortcomings like that. (T2_S2)
>
> *Mesela işte kısıtlamalar var App Inventor'da. Ama normal programlamada istediğini yapıyorsun. 1.yi kapat 2.yi aç 5. Yi aç falan diye şey yapabiliyorsun. Ama AI da o olmadığı için ve sınırladığı için seni mesela sürekli üstü üste birikmiş ekranlar yığını oluşuyor. Bu tipte handikapları var AI'ın. (T2_S2)*

Some students complained about the errors and crashes in App Inventor. Most of the students have afraid to lose their application due to some error or exit without saving, even though App Inventor has an auto-saving feature. As students T1_S7, T1_S11, and T2_S3 specifically explained their experiences regarding the crashes or technical shortcomings of App Inventor environment; there are still need for technical improvements.

> It does not save for example. It ended up bad for me. For example, the things I added into vertical, horizontal

> (arrangements) when I delete them it was all deleted. I had to make them all over again. (T1_S7)
>
> *Kaydetmiyodu mesela o kötü olmuştu benim için. Mesela bir şeyin horizontal, vertical onların içine attıklarım silince hepsi birden silinmişti. Onları tekrar yapmak zorunda kalmıştım. (T1_S7)*
>
> Once it scared me; I thought that I lost my application. Because of that I constantly downloaded after any changes I have made. (T1_S11)
>
> *Bi ara korkuttu beni; uygulamalarım gitti sandım. Evet onlar var. O yüzden sürekli bilgisayarda yaptığım herşeyden sonra değişiklikten sonra indiriyordum. (T1_S11)*
>
> If App Inventor will be enhanced a little, I think, better outcomes will emerge. Because, if an error or a crash occurs, it places us into a difficult position. Other than that, I do not think there is any shortcoming. The thing I complained about the most was problems I encountered in App Inventor. (T2_S3)
>
> *AI biraz iyleştirilirse daha iyi sonuçlar alınabileceğini düşünüyorum. Çünkü büyük bir hata olduğunda veya çökme olduğunda o bizi zor duruma sokuyor yoksa herhangi bir eksiklik yok benim en çok şikayet ettiğim şey AI da yaşadığımız sorunlar oldu. (T2_S3)*

Other than crashes and errors, the most common complaint was lack of undo feature. When the developer deletes a group of blocks or an arrangement of visual components, there is no turning back to the previous version, unless if he or she took a backup manually before as students T1_S5 and T1_S10 stated.

> There is no undo button. When you exit without saving, you lost all. (T1_S5)
>
> *Geri al yok. Kaydetmeden çıktığınızda hepsi gidiyor. (T1_S5)*

> Yes, for example, when you remove something, you lost all. (T1_S10)
>
> *Evet Mesela şeyden bir şeyi çıkardığın zaman hepsi gidiyo. (T1_S10)*

As it was mentioned in the previous code, students think that if an offline version of the App Inventor is provided it would be more convenient for them. The online version also restricts them to access from one place and open one project at a time. This type of restriction prevented students from benefiting from their old projects like looking at the codes as student T2_S6 stated.

> For example, if it were downloadable and usable like Scratch… You want to open the other one in the side window. Old example and new example side by side to look at it and make the new one. It lacks this (feature). (T2_S6)
>
> *Mesela şey gibi Scratch gibi indirilebilir ve kullanılabilir bir hali olsa… Şeyi açmak istiyorsan bir de yan tarafta açmak istiyorsun. Eski örnek burada yeni örnek burda olsun en azından bakarak yapayım o yok. (T2_S6)*

According to observation notes, technical problems (e.g. losing connection with the companion testing app) while testing the application made students think that they have a problem with their codes. Using a stable environment is important, but informing students about the probable problems could be essential for the first time learners.

Students also complain about App Inventor's getting slower when the application is getting more complex. According to the interviews, 13 of the 18 students complained about the slowness of the environment and their problems with that. According to the observations and students opinions throughout the course, all of the students complained about this disadvantage of the App Inventor Environment.

> I mean when we uploaded too many visual, video, or wrote too many codes, we face difficulties in App Inventor. It cannot render, or we constantly have to download the apk file to make it work appropriately… Maybe because, we wrote many codes

> but when we download it separately, loading it was slow. The reason could be that it is online. (T2_S3)
>
> *Yani, işte çok fazla mesela görsel video yüklediğimiz zaman çok fazla kod yazdığımız zaman AI da sıkıntı yaşıyoruz. Ya render edemiyor göremiyoruz ya da sürekli apk almamız gerekiyor düzgün çalışması için... Belki biz çok fazla kod yazdığımız için ama bize şey yaptığımız zaman ayrı olarak indirdiğimiz zaman yüklenmesi biraz yavaş oluyor mesela. İnternet üzerinden olduğu için olabilir. (T2_S3)*
>
> When block count is too many, there had been a slowness… Like we were back in Stone Age and worked there. (T1_S9)
>
> *Ama şey olayı bazen sıkıyor bu AI da program yaparken bloklar bayağı bir arttığı zaman hani bi kasma oluyor ya sanki taş devrine dönmüş de orda çalışıyomuş gibi oluyordu. (T1_S9)*

Student T2_S5 states that it was constantly getting slower and sometimes crashes after getting slower.

> Being slow… Sometimes it crashes. (T2_S4)
>
> *Kasması... Bazen Çöküyor ya. (T2_S4)*

Especially when testing their application through the App Inventor companion app, slowness of the environment was becoming a real problem since it causes lags as student T1_S1 and T1_S2 mentioned in interviews.

> It was getting slower. For example, my application is lagging 3 seconds (when testing). (T1_S1)
>
> *Yavaşlıyodu. Program evet şu an mesela benim programım 3 saniye sonradan geliyor. (T1_S1)*
>
> Negative sides… It was lagging. For example, there has been a change, but because I haven’t reset the companion, I could not see it. I realize that later. (T1_S2)

> *Olumsuz yanları geç geldi. Mesela yapıyodum ben değişiklik olmuş ama resetlemediğim için ben hala göremiyordum. Geç farkettim bunu falan. O biraz companiona öneri olacak da ne bileyim. (T1_S2)*

Environment's getting slower when students try to develop a more complex application was acting like a natural demotivator environment. It is also one of the main reason for preferring a textual environment instead of a visual one for the future.

Visual Flexibility Problem

Another point that students want it to be improved is the visual flexibility of the App Inventor environment. Students have problems while designing their applications visually. Using designer tab sometimes restricts students while designing the screen of the application. Especially, while they are dragging and dropping the components, they think that App Inventor was acting weird.

Student T1_S5 and T2_S3 emphasized a simple shortcoming nearly all of the students complained throughout the course. They were thinking that the visual design capabilities of App Inventor were very limited including the font library. They could not find the font types they wanted. Even it could be seen as an elementary problem; it affected their development process significantly.

> I could not use the font face I wanted. (T1_S5)

> *Yani istediğim fontu veremedim. (T1_S5)*

> Visually, buttons and images could be improved… Font type numbers could be increased for example. When I tried to add a sound or video, it caused some problems. Those can be improved. (T2_S3)

> *Görsel olarak butonlar koyulan resimler onlar belki daha iyileştirilebilir. Yazı tipleri arttırılabilir mesela kısıtlı fontların falan... Mesela şey geldi şu an aklıma ses eklemeye veya video eklemeye çalıştığım zaman o çok sıkıntı yaratmıştı projede. Onlarla ilgili bir iyileştirme yapılabilir. (T2_S3)*

Placing the components in the designer screen was also another technical shortcoming in App Inventor environment. Throughout the course, nearly all of the students had problems while assigning a component to a position on the screen. Students T1_S2 and T2_S4 expressed this issue as follows.

> The problem of the program (App Inventor) is that I cannot place them in anywhere I want without using arrangement. The designer made me angry and work extra. (T1_S2)
>
> *Programın problemi o arrangement olmadan istediğim yere koyamamam falan onlar sıkıntı çıkarıyor. Designer kızdırdı yani biraz uğraştırdı kızdırdı da yani. (T1_S2)*
>
> We cannot place the thing where we want. When we work on designer screen, it can be freer. (T2_S4)
>
> *İstediğimizi istediğimiz yere koyamıyoruz. Şey anlamında o designer ekranında çalıştığımızda biraz daha free olsa. (T2_S4)*

Another tab (or screen) that App Inventor has is Blocks tab. Blocks tab is where the developer write (or create for App Inventor case) the code. It is just a blank screen where the developer could drag and drop the blocks anywhere her or she wants (figure 4.9). However, the screen is an infinite scrolling area. When the screen is full of blocks, the developer needs to scroll to an empty place. Students also find this inappropriate for the big scale projects. They used the zoom feature of the browser to overcome this problem. However, this minimizes the blocks up to a scale where they are unreadable. As the researcher was writing of this dissertation, MIT updated the App Inventor with a zoom button placed at the blocks screen to overcome this problem.

> Blocks area is too narrow, I mean, OK, we did not make anything complex, but still, we wrote some. Because of that, we constantly shrink and enlarge the screen. (T2_S6)
>
> *O blokların olduğu kısmdanki yer çok küçük mesela hani tamam biz çok komplike bir şey yapmadık ama yine de sonuçta bir şeyler yazdık yani. Onun için şey oldu sürekli biraz küçülteyim biraz büyüteyim. (T2_S6)*
>
> It is very hard to go to the bottom of the screen. (T2_S7)
>
> *Mesela ekranda en alta inmek çok zor. (T2_S7)*

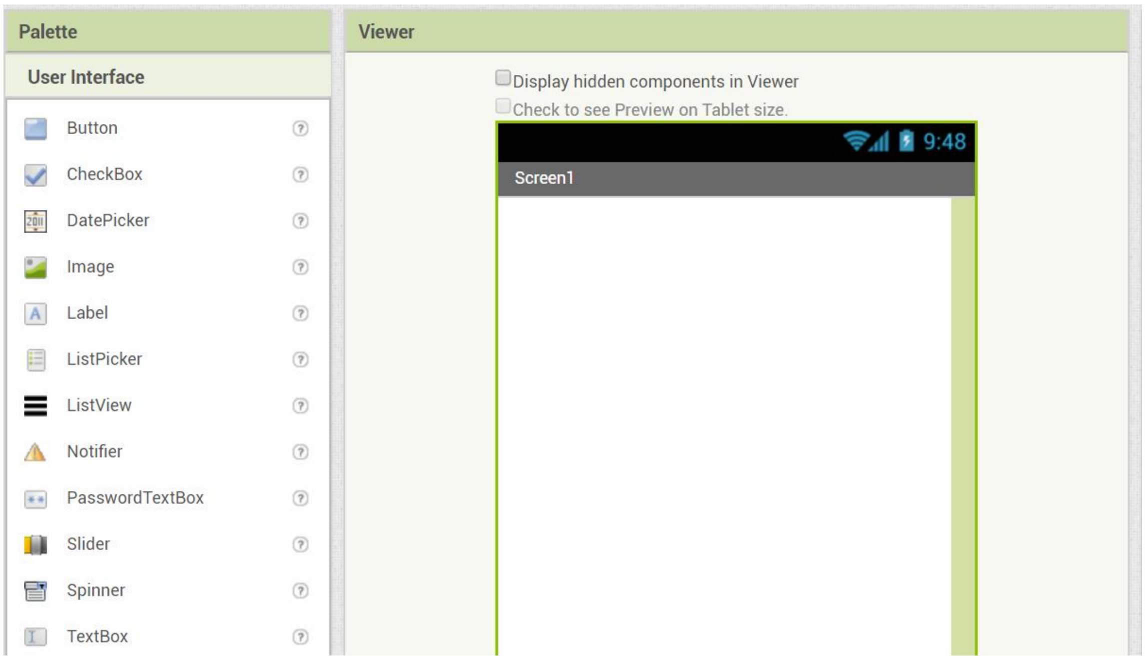


Figure 4.9 App Inventor Interface

Summary of negative aspects

Similar to positive aspects students also reported negative aspects of the App Inventor environment. Mostly mentioned negative aspect was that App Inventor is not flexible enough for advanced programming. Especially students with programming background or with programming career goal found App Inventor inadequate. They put forward that students with a programming career plan should learn a textual programming language like C++. In addition to lack of advanced features, students also criticized the crashes and errors they have encountered during their project development. Moreover, students also reported that the more complex the application gets, the slower it becomes. Some students linked those errors and slowing to being solely online.

In addition to programming problems, students also complained about the lack of visual design features. They could not find enough fonts, they have had problems placing and moving objects etc. Another mostly mentioned lack of feature was not having common software features like undo and redo. It could be seen clearly that as the App Inventor environment and similar environments are developed more maturely, it would be embraced more by the non-programmer audiences.

4.4.4 Summary of Programming and Programming Environments

Some of the important points emerged under themes will be emphasized under this part. In addition to examination of the advantages of visual and textual programming use, specifically positive and negative sides of the App Inventor environment will be listed.

- Attitude change towards programming
    - Visual Environment helped students to overcome their fear towards programming
    - Being able to develop a working product increased their self-confidence regarding programming
- Comparing textual with the visual programming environment
    - Advantages of using only visual programming environments:
        - Suitable for first-time learners
        - Product-oriented / Concrete outcome
        - Instant development and immediate feedback
        - More practical / More chance of implementation
        - No syntax error

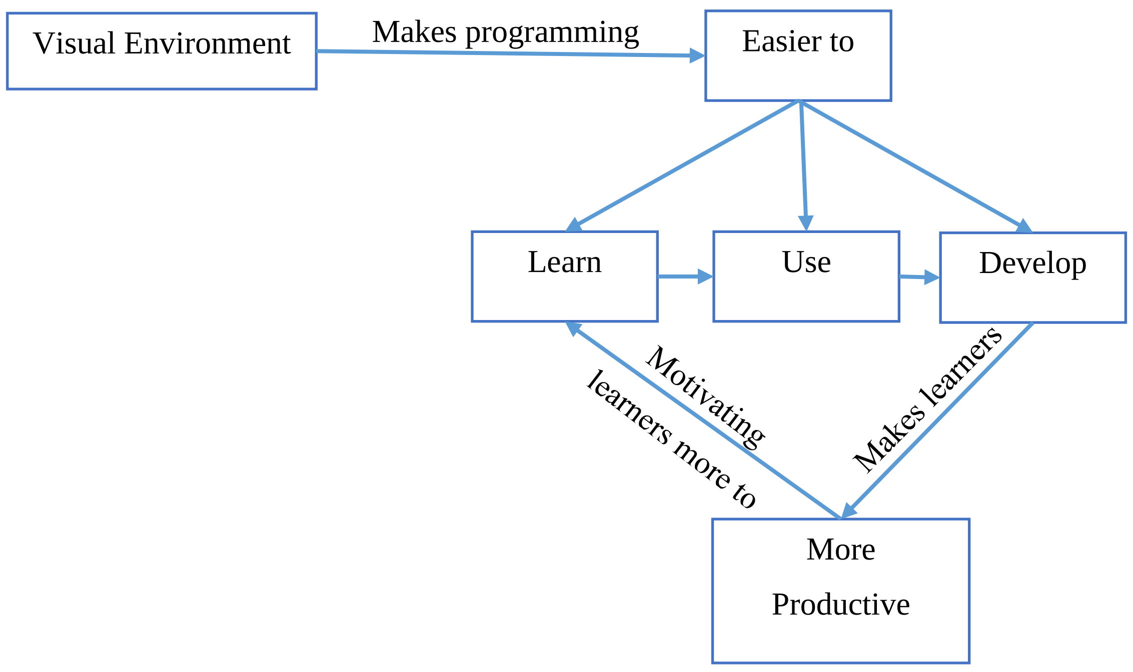


Figure 4.10 Effect of Visual Environment on Novice Programmers

        - Advantages of using a combination of both environments
            - Benefits of the visual environment
            - Knowing a different and more flexible environment
            - Preparation opportunity for advanced programming
            - Attitude change towards textual programming
        - Advantages of using textual environments
            - More flexible and advanced environment
- Positive sides of App Inventor environment (similar to advantages of visual programming environment)
    - Visual structure of App Inventor was easy to learn, use and develop
    - Have helpful interface and arrangement
    - Immediate feedback and testing without compiling
- Negative sides of App Inventor environment
    - Not flexible enough for advanced programming
    - Technical deficiencies: crashes, errors, lack of basic features, online-only software
    - Visual flexibility problems: lack of font types, visual design problems
    - Getting slower when creating complex applications

4.5 Dynamics and Evaluation of the Course

Students were also asked to evaluate the course, tutorials and the general process of the course. In addition to the evaluation of the course, the programming concepts that they find challenging, and the changes that they think beneficial, was also taken into consideration to reshape the course. Unlike other themes, under this theme chronological order of the interviews will be followed to form a clearer picture of the process and the outcome of the course.

4.5.1 Challenging Concepts of Programming

4.5.1.1 Variables

One of the essential programming concepts that were found hard to understand by students was the variable concept. The first example with variables was presented to students in the third week of the course. After the first example, many examples with variables were presented. Students stated that particularly at the beginning of the course variables in programming was confusing and difficult to understand. During the course hours, some students called the variables as “orange things” since the variable blocks were in orange color. That word turned into a joke for them, representing that they do not understand how to use variables in their applications. In addition to the observations, the students also emphasized that during the interviews. While student T1_S9 mentioned the variables as their nightmares, T1_S2 was thinking that understanding variables was causing some problems.

> Our nightmare was orange things. (T1_S9)
>
> *Korkulu rüyamız turuncu şeylerdi. (T1_S9)*
>
> Variables were problematic in my opinion. (T1_S2)
>
> *Variablelar bence problemdi. (T1_S2)*

Student T1_S11 explained the essence of the problem as using the variables. He stated that he did not have difficulty understanding the definition part of the variables, however using the variables was the real issue for her.

> Orange things. Variables. After defining, I mean, OK, we define it but after defining, using it… (T1_S11)
>
> *Turuncu şeyler. Değişkenler. Değişken tanımladıktan sonra tamam tanımlıyoruz ama o değişkenin nasıl kullanıldığını… (T1_S11)*

Students from Elementary Mathematics Education have some additional problem understanding the variable concept since they use variables in their own profession in a different way. Having the same name, but different use led to a misconception and made them need some special attention to understand the concept. Student T1_S3 states that they always use the variables in mathematics, however defining the variables and changing them through the application was causing some misconceptions for them.

> Because of the variables, I mean we use them in mathematics, but when it comes to this, what will we say, we should define it. That’s why it’s different… It was really different for us. I

mean, what will we define? Because, you know, we decide the variable on our own, probably because of that. (T1_S11)

Çünkü değişken hani matematikte kullanıyoruz ama buraya gelince değişken ne diyecez kendimiz belirleyecez ya o yüzden farklı oluyor… Bizim için evet gerçekten farklıydı. Yani ne tanımlıycaz. Çünkü değişkene kendimiz karar veriyoruz ya biraz o yüzden sanırım. (T1_S11)

Student T1_S1 stated that he did not understand variables at the beginning, using variables in examples regularly, made him understand the concept.

Variable ları başta anlamamıştım. (T1_S1)

I didn't understand the variables at the beginning. (T1_S1)

Student T1_S11 recommended using basic applications to learn the concept then continue to use the variables in tutorials that are more complex could be better in terms of understanding the variable concept. The suggestion of this student was used directly in the course which was more successful than the predecessor course.

Just to show how variable could be used, a basic application could be made. And then we could add that (information) into an application; it could be more efficient. It could be easier to understand. (T1_S11)

Evet sadece değişkeni bu nasıl kullanılıyor onla basit bi uygulama yaptıktan sonra bizim diğer bir uygulamanın içine katılsaydı daha verimli olabilirdi anlaması kolay olabilirdi. (T1_S11)

Based on the observation of the instructor, there was another part that was hard to grasp by students regarding using variables and other values specific to App Inventor environment. To define and give a value to a variable, users need to use "set" block; and to use the variable they need to use the "get" block (in figure 4.11) below from top to bottom, defining a variable, assigning a new value using 'set' block, and giving the value of variable to a label using get block). Some of the students find it confusing since it is not using any equality system. They were making mistakes about when to use set and when to use the get block. Very similar confusion was observed more than one time about the general structure of blocks. Students were choosing the block with the name of the component (e.g. textbox1), instead of using the block which contains the data (e.g. textbox1.text). At the first class students with previous programming experience had problems with no-tutorial examples. Structure of the blocks and the variables should be verbally explained to the students. Even the ones with previous knowledge of programming since they could have a misconception about the using the variables in the visual programming.

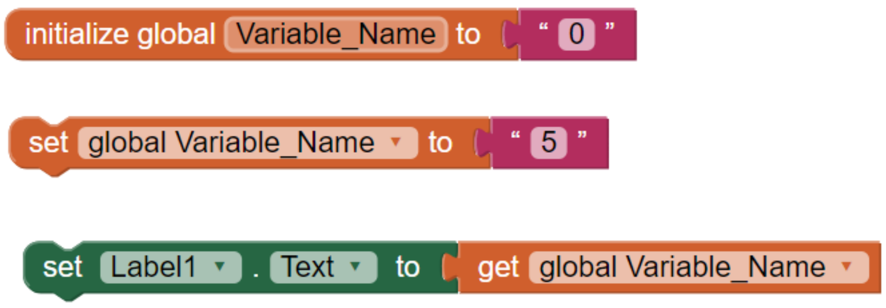

initialize global Variable_Name to " 0 "
set global Variable_Name to " 5 "
set Label1 . Text to get global Variable_Name

Figure 4.11 Defining and Using Variable in App Inventor

Another data extracted from the observation notes was that students were confused about the mathematical variables and programming variables. When variables name was used students saw it as the mathematical variable which represents a number or involves the numerical value. Especially assigning and constantly changing the value of the variables confused the students. One possible solution for this misconception could be explaining the differences between the variables in programming and mathematical variables, at the start of the variables topic.

4.5.1.2 Clock

Students also found the clock concept as difficult to understand. The clock was used as an alternative loop method in the course and generally used in-game examples. As student T1_S2 stated that he had some difficulty while trying to understand the clock concept and its components. He indicated that he understood the concept and its properties after recurring examples of the clock.

> For instance, I had a hard time understanding the logic of clock for a while, but I understood it at the end. What interval is, what enabled is… (T1_S2)

> *Mesela şu clockun mantığını anlamakta biraz zorlandım ama anladım sonunda. Interval neymiş enabled neymiş. (T1_S2)*

Even the students with programming background have not used components similar to the clock. Time concept was different for them since the examples they have made were mostly output based. Even, student T1_S4 who was an intern PHP programmer thinks that clock concept is hard to understand. He stated that he needed to study the clock at home.

As in the other difficult concepts, it could be also beneficial for clock concept to familiarize the students towards the concept with as simple as possible. Student T1_S2 suggested having a countdown timer example for clock concept which was applied in the second term. However, students still have difficulty at the beginning of clock concept.

> I, too, had difficulties in Clock (component). This was my first time using a Clock (component). (T2_S3)
>
> *Clockta ben de sorun yaşadım. Clockta ilk defa çalıştım. (T2_S3)*

T2_S6 stated that the first activity he found hard was clock concept. He thought that it was similar to the loops. However, he was not sure about how it works. Students with previous programming knowledge tried to make a connection with their old knowledge about the similar concept. So, it could be better to make the connection more apparent to them.

> For example, first activity I've had difficulty was… Different than C++, I mean on the other software there was loop concept, but there wasn't any time concept. I mean, we didn't use any. We have used the loops, and we haven't even seen how they were working. That gave me some hard time, first. (T2_S6)
>
> *Etkinlikler ilk şey zorlayıcı mesela. C++tan farkı da hani Diğer programlamalarda şey kavramı yok hani döngü kavramı var ama zaman kavramı yok hani onu hiç kullanmamıştık. Döngüleri kullanıyorduk döngülerin de hani işlediğini bile görmüyorduk. Ilk başlarda da o beni çok zorlamıştı. (T2_S6)*

Although T2_S6 was having difficulty understanding the clock first, he had effectively used three clock components in his project. He also looked at the projects of the other students to understand how they used the clock component and used their projects as a source of knowledge.

> I didn’t understand the Clocks first, but, afterward, I started to understand those, while I was making the game (project). (T2_S7)
>
> *Clockları en başta anlamamıştım ama sonra onları anlamaya başladım sonlara doğru oyun yaparken. (T2_S7)*

4.5.1.3 Database

Both students with advanced programming knowledge and novice programming learners had a hard time in understanding the database. While student T1_S4 stated that he was confused at first when he was introduced to the database in App Inventor, student T2_S7 declared the database as the hardest topic.

> Database topic quite confused me. That could be elaborated in detail. (T1_S4)
>
> *Database konusu bayağı kafamı karıştırdı. Onun üzerinde daha detaylı durulabilir. (T1_S4)*
>
> The hardest part was the database in my opinion; others could be understood in one way or another. (T2_S7)
>
> *En zoru database bence diğerleri bir şekilde anlaşılıyordu. (T2_S7)*

Student T2_S6 stated that he understood the database concept. However, he did not learn the concept fully. He used the older examples to overcome his lack of knowledge

> I understood the database when you first explain it to us, but I did not comprehend it fully… I went back frequently and looked at the examples we made like the walking example. I looked at it to see how we used the images, how we used the database… (T2_S6)

> *Database'i ilk anlattığınız zaman anlamıştım ama tam oturmamış… Onlara ben sık sık dönüp baktım yani şey yaparken mesela yürüyüşü nasıl yapmışız resimleri nasıl kullanmışız orada database'i nasıl kullanmışız... (T2_S6)*

According to the observations and the interviews, students needed more examples and more emphasis on the database concept. Students also reported that revisiting the examples they have completed in course hours helped them to understand the concept. It was clear that providing an open and online library of examples to students is essential as it was once again mentioned by the students.

4.5.2 Course Dynamics

This course was designed as an introductory programming course, which uses App Inventor Environment to teach the programming concepts to the learner with real-world examples. The course was an 8-week long elective course, open to all departments within Middle East Technical University. For two semesters, the same course was given under Computer Education and Instructional Technology. Since the research framework of this course was design-based methodology, the design of the course reshaped based on the feedback from students. In addition to the observations and opinions of the students throughout the course, at the end of each semester, by conducting interviews with students some key points and guidelines were detected. Those key points and the guidelines could shed light on the fruitful and incompetent parts of the course.

4.5.2.1 Discovery Learning – Reshaping the Tutorials

One of the most common feedback taken from students regarding the tutorials was after the first couple of weeks they need less explanation more discovery and problem-solving. While the course was initially designed as a self-learning course through tutorials with the instructor as a guide, too much explanation and well-structured steps stopped students to think about the example that they were working on. Since the programming environment was visual and tutorials was showing every step and every block which were needed to be created, students stated that after some point, they started to imitate the same image they were seeing in tutorials without any effort and thinking.

Fewer Images in Tutorials

Student T1_S5 remarked that tutorials with exact images of the blocks screen make them copy and paste the exact images in their application without any effort and thinking.

> I mean, we were copying and placing the exact image that we see, maybe that could be a problem regarding the learning. (T1_S5)
>
> *Yani birebir o resmi kopyalayıp yani gördüğümüzü oraya koyuyorduk belki o yönden dediğiniz gibi öğrenme yönünden sıkıntılar olabilir. (T1_S5)*

As student T1_S3 specified that students were not actively thinking the solution while using the tutorials, he suggested that rather than tutorial based class, basic applications could be started to be made.

> Because we always look there and make it. We weren't thinking much about it. Maybe, instead of making many tutorials and then starting to make applications, they can be put into the middle. (T1_S3)
>
> *Çünkü ordan hep bakarak yapıyorduk üzerine çok düşünmüyoduk sanırım. Belki şey olabilir bir sürü tutorial yapıp da sonra birden uygulamaya geçmek yerine araya konulabilir. (T1_S3)*

T1_S9 thinks that the tutorial with images of every step including the blocks stops them to think and just matching the image in the tutorial with their application in App Inventor.

> I have seen that many of my friends and me also, we look at the picture, look at the blocks. After some time it turns into matching… The ones that you have prepared could be in detail for the first and second applications, but after those, text without picture could be used. (T1_S9)
>
> *Bir çok arkadaşta şeyi gördük ben de mesela yaptığım oldu. Resme bakıyoruz bloklara bakıyoruz. Eşleştirme gibi oluyodu bir süre*

> *sonra. Sizin hazırladığınız tarzdaki 1-2. Uygulamalar için olabilir ayrıntı olayı ama sonrasındakiler için şey yapılabilir mesela sadece resim olmadan metin. (T1_S9)*

The More Practice, the Better

This problem was detected before the interviews. After the second week, instructor added “in-class assignments” with no tutorials. In-class assignments were simple problems with just screenshots of the end-product without any steps. Students were confused at first when they think that they were successfully completed the tutorials of the first two weeks. They found the basic example more difficult since there was no image of the blocks screen. At the end of the course, even though, they found them more difficult, they thought that they are beneficial and there should be more of them. Student T1_S3 was also thinking that the more practice, the better in the time of her interview.

> We could have developed more applications. On the other hand, we developed a bunch. Since we have never taken anything (course) like this, the more practice, the better for us. I mean that could be nice, and we were making them (applications) from the tutorials. We were confused when we jumped into application making part. (T1_S3)

> *Daha çok belki uygulama geliştirebilirdik ama zaten bayağı bir uygulama geliştirdik. Biz daha önce hiç böyle bir şey almadığımız için ne kadar çok pratik yaparsak bizim için o kadar faydalı olacaktı. Hani o güzel olabilirdi bi de hani şey düşünüyorum tutoriallardan yapıyorduk ya ordan bi anda kendimiz uygulama yapmaya geçince biraz şaşırdık. (T1_S3)*

According to observation notes, students ran into a stone wall when they encountered problems without tutorials. So, it could be better to use instant demonstration, feedback, guidance, and explanation of the instructor for the first couple of in-class assignments (problems without tutorials). As it was recommended in the first principles of instruction, explaining the complex steps with demonstration was used in order to help students understand the concept or procedure. However, in the following weeks, they adapted to the solving process by using some strategies. Some students get help from each other and instructor to understand what the problem is. Some of them linked the current problem with previous examples. Remembering students to thinking step by step, and computational thinking which was mentioned in the theoretical hours also worked for some of the students. Even though examples without tutorials took much more time than the examples with tutorials, students got used to solving problems by using computational thinking. Observation notes also showed that in examples without tutorials, some of the students tried alternative solutions and asked for the approval of the instructor to go further with their solutions. After they have regular examples without tutorials and homework, they learned to try and test the solution. To help them through the baby steps phase, it could be beneficial to use training wheels-like problems with prepared steps for the first two weeks of the problems with no tutorial. It could help students to see the next step and embrace the computational thinking.

Removal of Tutorials after Two Weeks

Student T1_S2 thinks that if there were fewer tutorials and more discovery, students would enjoy more. He thought that after the first two weeks tutorials could be removed. While tutorials were beneficial to build the initial knowledge and support the new learners' motivation towards the programming, students demand freer area to discover by themselves. The important point of this statement, which other students agreed upon, even the students with little knowledge need to explore and discover the learning by themselves.

> Nevertheless, as I have said before, if tutorials become less, first 1-2 weeks using tutorials then remove the tutorials, in my opinion, students would enjoy more. It would be better to ask the next generation, but I like to learn by exploring, that would be more interesting for me. (T1_S2)
>
> *Ama dediğim gibi tutorialler az olursa ilk 1-2 hafta tutorial üzerinden gidip ondan sonra onları kaldırsak bence öğrenciler de keyif alır. Gelecek nesillere de sormak lazım da ben keşfederek öğrenmeyi sevdiğim için daha çok ilgimi çekerdi benim. (T1_S2)*

Thoughts of the student T1_S9 were similar to T1_S2, well-structured steps are good for the first week. After that, in tutorials, less detail should be provided, fewer images of blocks should be used, and more textual guidance should be given to make them think more. He also thinks that in-class assignments with no tutorial or guidance are also beneficial for them.

> Too much detail were given in some of them. I mean, like “you must do this like that”. In my opinion, that should only be in the first one. First one was nice. I like the first one very much. The application after that makes us think a little more. The one with just the text… Like “you should do this, create a canvas, throw the ball and make image sprites do this”. Those parts, I don’t know, we make that one with more thinking. (T1_S9)
>
> *Bazılarında çok ayrıntı veriyoduk ya hani mesela bu böyle yapılmalı bu böyle yapılmalı bence o sadece ilkinde olmalı ilk güzeldi çok hoşuma gitti benim. Sonrasında şey uygulaması mesela biraz daha düşündürmeye başlamıştı; Hani sadece metin vardı. Şunu şunu yapmalısın mesela canvas oluşturmalısın. İşte top atmalısın imagespritelarla şunu yapmalısın falan gibi kısımlar olduğu zaman ne biliim biz biraz daha düşünüerek yapmıştık onu. (T1_S9)*

T1_S2 mentions a solution to overcome the match the image problem. He thinks that students should work on solving a problem in course hours. Well-structured tutorials should be available after course hour to be checked by students to see the solution.

> Hani önce bi uğraşsın ama kendi. Direkt olunca çünkü açıp bakıyoruz belli bir şeye. (T1_S2)
>
> *I mean let them (students) work on it first, when it is right there, we look at it (tutorial/solution). (T1_S2)*

Student T1_S6 states that they can learn the application just by discovering and trial and error. According to her, the instructor should explain the example and the blocks in general, and he should let students try them by themselves. T1_S7 also thinks that when she learned by doing it rather than following the tutorial step by step, she learns better.

> I think it is enough that you give a lecture in the beginning. After that, it is like a trial and error type of software in my opinion. I mean, you can learn by trying. (T1_S6)
>
> *Bence işte sizin başta anlatmanız yeterli olacaktır diye düşünüyorum. Sonrasında zaten biraz da deneme üzerine kurulu bir program gibi geliyor bana. Yani deneyerek öğrenebiliyorsun. (T1_S6)*
>
> When I learned by doing, it was more effective. I learned what makes what. If I understood the function at the beginning, instead of drag this to here, drag that to there, it would be better. (T1_S7)
>
> *Kendim yaparak hani öğrendiğimde daha verimli oldu. Neyin ne işe yaradığını öğrendim. Öyle başlangıçta hani böyle bunu buraya sürükle bunu buraya sürükle yerine ne işe yaradığını anlasaydım daha güzel olabilirdi. (T1_S7)*

When students started to do their own project, there was not any tutorial or guide in their hands. With a bigger goal to complete, it made them think more than the in-class activities or homework. Student T1_S11 stated that when she started to do her project, she needed to look at the blocks and their functions again to understand why and how they work. This could be a proof of open-ended, and bigger assignments are beneficial for students to learn by doing and discovering.

> Daha sonrasında tek tek bakmam gerekti aslında bu hani buraya neden buton olmuyodu label oluyo. Neden textbox olmuyo da o oluyo şeklinde daha çok düşünmemi sağladı. (T1_S11)
>
> *After that (project started) I need to look at them one by one. I mean, why the label is used instead of a button. Like why textbox could not be used but the other could be… It made me think more. (T1_S11)*

Since nearly the half of the students from the first term suggested that tutorials should be removed, in second term each week an example without a tutorial was given to students. However, observations showed that the students had problems with the basic examples without the tutorials. They checked and tried the blocks one by one to solve the problem, even for an easy example as finding the sum of two different variables. Therefore, removing the tutorials would be confusing and difficult for the students. Providing different kinds of examples, and finished examples with missing pieces as it will be mentioned in the blending the top-down and bottom-up approach part, could be supporting for the students with different learning style and knowledge level.

Dynamic Tutorials and Develop-it-more activity

Student T1_S11 suggested a strategy to overcome the tutorial problem. After every week, students should make the last week's example without looking at any tutorial or resource. However, making the same example could be boring for students who have enough knowledge about the topic and could be challenging for students with less knowledge. Instead of that instructor added a challenge at the end of each tutorial to let them make their own version by developing it more. Develop-it-more activities let them enhance the application without using any tutorial in which challenges were offered students to complete after finishing the tutorial. According to the observation notes from the first week of the first term, two students who have previous knowledge completed the tutorials without any excitement or being enthusiastic about it. Those activities did help the students with previous knowledge about programming in the following weeks by providing them challenges after completion.

All of the students liked the in-class activities including tutorials. However, even in-class assignments without any tutorials were provided to the students in the second term; they still think similar to the students in the first term. As students T2_S3 and T2_S5 suggested, tutorials should have missing steps based on the knowledge of the learner. According to their recommendations, tutorials would be better if they are dynamic based on the knowledge level of the student.

> In some tutorials, what we should do was written step-by-step. Maybe that could be up-to-us. That simplifies it. Of course, there are people who never take the course. (T2_S3)
>
> *Bazı tutoriallarda adım adım ne yapmamız gerektiği yazıyordu ya o biraz bize bırakılabilirdi belki. O biraz basitleştiriyor ama tabi hiç almayan insanlar da alıyor dersi. (T2_S3)*

> Activities were fun. It was both informative and entertaining, and each has a different feature. That was nice. Sometimes I think that, I mean, what if not all of the steps were given. Make us think, make us… Force us, I liked that type of activities more, frankly. (T2_S5)
>
> *Etkinlikler gayet eğlenceliydi. Hani hem öğreticiydi hem eğlenceliydi her bir etkinlikte farklı bir özellik vardı. O güzeldi. Bazen şeyi düşünüyorum hani bütün adımları vermese mi diye. Direkt bizi düşündürsün bizi şey yapsın. Zorlasın, o tür etkinlikler daha çok hoşuma gidiyordu açıkçası. (T2_S5)*

On the contrary to the other students in second-semester, student T2_S7 found tutorials helpful but have difficulty while making the in-class assignments without any tutorials. T2_S7 was the only student in the second semester with no previous programming knowledge. So, instant guidance should be provided for students who are having difficulty

> I think the tutorials were very good. Others were giving me some hard time. (T2_S7)
>
> *Şeyler bence çok iyidi mesela tutoriallar onun dışındakiler beni biraz zorluyordu. (T2_S7)*

According to observation notes, students from computer education and instructional technology program found the tutorials easy and completed faster than other students as expected. Students from different backgrounds or different interests areas learning pace could be different. Designing flexible examples could help them to stay in the flow of the course. While giving some ideas to students could be helpful, letting students find their own ideas to develop the application further would also be beneficial. Having an audience from different fields and students with different knowledge level could cause problems regarding the pace of the course. Student S2_T2 who has advanced programming knowledge thinks that course could be a little faster, on contrary to his classmates, because he was finishing the examples and additional develop-it-more activities in class. So, if the class is more diverse regarding previous knowledge of the students, the course could be more flexible for students with no knowledge and with a programming background.

> It could be a little faster. Even if my classmates told that we are going a little faster, I think that’s because they threw back a little in the beginning. I mean you could increase the speed a little. (T2_S2)
>
> *Birazcık sadece hızlanabilir. Benim şeyim her ne kadar sonuna doğru birazcık hızlı gidiyorsunuz dese de arkadaşlar onlar bence baştakileri aksattığı için o duruma düştüler. Yani birazcık hızı yani tempoyu arttırırsanız. (T2_S2)*

Observation notes from the first week of the first term put forward that students who had finished the tutorials looked for ideas to develop the applications further than the tutorials without any encouragement. After that week additional challenges were presented students at the end of each tutorial. It was called develop-it-more activity which was used to show a way to students to let them go further with the example. Observation notes showed that providing challenges also motivated students to develop the application of the wee even more. Providing additional missions/challenges could be an effective and motivating strategy for the faster students. On the other hand, students from the second term finished the first examples and started improvements without any encouragement or reminder. If develop-it-more strategy would be a tradition for students, they will continue to search for ways to improve their applications further.

The instructor provided step-by-step tutorials (most of them were from MIT’s website) which include fundamental concepts of App Inventor and programming in general. Some students suggested that tutorials could be removed to provide them a freer area for discovering the program. However, removing step-by-step tutorials and let novice learners deal with new information just by the discovery in a complex area like programming could be confusing and demotivating. Rather than removing the tutorials, it would be better to manipulate the tutorials into the ones with missing steps, which could keep the attention of the students. Every tutorial should include challenges regularly to keep the attention of the students, let them explore the solution by themselves, and at the end, there should be an open-ended task at the end to provide a way students to proceed even further. After the feedbacks of students in the first semester, both a tutorial and a challenge were provided.

Homework

In the first semester, only two homework were given to the students. Students of the first semester found given homework as very beneficial for them. They think to improve the course more homework should be provided for the students. They believed that homework is essential to understand the logic of the programming. Moreover, as students suggest that tutorials should let them learn by discovering, they thought that homework is also helping them to learn by discovering. Since they were thinking homework was beneficial, they suggested that more homework should be provided as student T1_S1 did.

> But, for example, 1 homework per week would not tire us. (T1_S1)
>
> *Ama mesela haftada 1 ödev yormazdı bizi. (T1_S1)*

Student T1_S7 also emphasized that she need some additional homework to understand the logic. Student T1_S11 also highlighted that homework helps them to exercise of new concepts they have learned.

At the beginning of the first term, the instructor gave students tutorials in which every step of the application was taking part. After the interviews of the first term, it was revealed that students prefer to discover some steps by themselves. They want to solve the problems by themselves. In addition to tutorials, students from the first term suggest that more homework could be beneficial for them to learn by discovery as T1_S2 stated. Student T2_S4 from the second term approve the benefit as he states he liked the homework since he needs to discover the solution by himself.

> Like the Street Fighter example, when we work on it on our own, I enjoyed more, to be honest. In the end, I discover something when I worked on it. (T1_S2)
>
> *Şu Street fighter falan olsun kendimiz uğraşınca daha çok keyif aldım ne yalan söyliim. Sonuçta bir şeyler keşfettim yani uğraşırken. (T1_S2)*

> Specifically, ourselves, without knowing anything… I mean, before showing us you gave us the homework, we discover how to do the homework. Those parts were fun for me. (T2_S4)
>
> *Spesifik olursak, kendimiz hani hiçbir şey bilmeden… hani bilmeden derken ödev veriyorsunuz ya mesela hani nasıl yapılacağını falan kendimiz buluyoruz o kısımlar benim açımdan zevkliydi. (T2_S4)*

While students from the first term were demanding more homework, some students in the second group think that there was too much workload. Since the course was updated based on the feedback from the students, nearly every week students have had homework. Student T2_S3 stated that the workload of the course was a little more than she expected since the course is elective. Student T2_S1 and T2_S6 also were thinking that having a homework every week was a negative side of the course.

4.5.2.2 Blending Top-Down and Bottom-Up Approach

Another suggestion from the students that was not tried by the instructor was blending top-down, and the bottom-up approaches for students to let them explore and understand the function of each block. Top-down approach starts with the big picture instead of building the program form the start. Normally bottom-up approach was implemented in the course. However, students suggested the use of top-down approach in addition to the bottom-up approach without knowing the names for them. Student T1_S10 exemplify the strategy she was thinking of as completing the missing part of the created application.

> If there were… For example, we saw an incomplete part of a program. We can complete that part. I mean, after we improve our self, we can do that by ourselves. (T1_S10)
>
> *Hani şu da olsaydı mesela bi programda bir şey eksik görüyoruz. Onu hani kendin tamamlayabilirsin. Yani kendini geliştirdikten sonra kendimiz tamamlayabiliriz. (T1_S10)*

Student T1_S11 also suggested a similar tactic to enhance learning. She was thinking trying blocks in a complete program to see what is the function of each would be more beneficial for her to understand the difference between blocks. The same student had offered the same tactic while taking the course.

> For example, I prefer to learning like this, I mean, by looking at the whole. Because when you make a change, you see the difference. Like "is that one compatible with here"… Learning could be more beneficial when step by step by adding the blocks one by one. (T1_S11)
>
> *Mesela ben böyle öğrenmeyi tercih ederim hani bir bütüne bakıp. Çünkü hani yaptığınız her bir değişikliği görüyosunuz bu burda oluyo mu böyle hani aşama aşama blok ekleye ekleye öğrenince daha yararlı olabilir bence. (T1_S11)*

Student T2_S4 also suggested using non-working examples with missing parts to complete. He suggested that students can complete the codes and make the application work.

> There could be a missing one near to the end. Even a person with no knowledge could do that by looking… If we were trying to find the non-working part, it could be more informative. (T2_S4)
>
> *Eksik verilebilirdi sonuna doğru belki direkt hani hiç bilmeyen birisi bile bakarak apabilirdi onu… Neresinin çalışmadığını bulmaya çalışsak daha öğretici olabilirdi. (T2_S4)*

Observation notes also showed that some students want explanation of each block instead of learning it from the tutorial. Before implementation of the block through the tutorials, using simple finished examples with missing pieces could be helpful for the students to let them see the difference that block makes to the program. Blending top-down and bottom-up approach could be an effective strategy for the first time learners.

4.5.2.3 Theoretical Hour

The course was mainly focused on implementation and creating examples since the researcher was following the spiral approach. During the first term, theoretical hours was used to explain the theoretical side of the programming and to show the textual simplified equivalent of the programs we made in the lab hour. Additionally, flowchart diagrams and basic algorithms were also presented to students. However, observation notes displayed that even the students who were very active in lab hours were quiet and unwilling to participate in the theoretical hour. When there was an application, students became more active during the course. Another strategy to make students more active could be giving students on paper activities like flowchart challenges or CS Unplugged activities. Flowchart challenges or CS Unplugged activities should be directly related with their daily routines or the applications from the lab hours to motivate them towards the activity as it was mentioned before.

Another problem was the difficulty of connecting the theoretical background with the application part. Researcher tried to connect the theoretical hours and lab hours through the applications student made in the lab hours. Demonstration tactic of first principles of instruction was followed in order to give students a clear understanding of each block and function and its algorithmic background. Demonstration was also used in the lab hours after the hands-on tutorials. It was helpful for students to have a clear understanding of the function of each block. Carrying that tactic to theoretical hour made it even more effective for students both for being active in the class and retaining the knowledge acquired from lab hours.

Feedbacks of the students from the first term was used to overcome the problems and reshape the theoretical hour into a more effective one. After the changes according to the feedback from the students through interviews and observations showed that use of theoretical hours as the recitation hours for the examples, demonstration of the blocks and the new concepts of the week was as important as the implementation by itself after the improvements and updates.

Verbal Explanation / Recitation Hour

Student T1_S2 stated that it would be beneficial to explain the function of each block that was used in the tutorials. According to him, that would help them to understand the parts that they have superficially followed through the tutorials. Student T1_S11 was also thinking explaining the features of a block could be more beneficial regarding understanding the block completely.

> I mean your explanation on the blackboard is more beneficial than the tutorials. Because, when we see the picture (in tutorials), we do not think. (T1_S2)
>
> *Yani sizin orda tutorialden daha fazla faydalı oluyor siz tutup tahtada anlatıyorsunuz şu şöyle bu böyle o çok daha faydalı oluyor bizim için. Resmi görünce düşünmüyoruz çünkü biz. (T1_S2)*
>
> After I started to be able to develop something, I started to enjoy more. After I started to understand more that which block does what… Maybe you can give us something about that one (block) and show us the features of it and then we could make the activity there (tutorial). (T1_S11)
>
> *Ben de bir şeyler yapabilemeye başladıktan sonra daha çok zevk almaya başladım. Daha çok anlamaya başladıktan sonra hangi blok ne işe yarıyo… Belki hani ilk başta bize siz onunla ilgili şeyler verip özelliklerini gösterip ondan sonra herhangi oradaki bi aktiviteyi yapsak. (T1_S11)*

Student T1_S3 also put forward a similar idea as a recommendation for the course for the theoretical hours. She stated that theoretical hours could be used as a recitation hour to recap the previous week's example and homework. In addition verbal explanation of the function of each block.

> Actually, I find the theoretical hour very beneficial. For example, we developed an application, something with a car, the first application we made without a tutorial… First, I was confused about how to do it by myself, I close the gap by coming the theoretical hour… In addition, if we learn stuff like which block does what, in theoretical hour, it would be more beneficial. (T1_S3)
>
> *Aslında teorik saati ben faydalı buluyorum örneğin bir tane uygulama geliştirmiştik araba üzerine ilk kendimiz tutorial olmadan yaptığımız uygulama. Araba böyle hareket ediyodu ekranda. Onu ben kendim başta çok şaşırmıştım nasıl yapacağımı onu teorik saate*

> *gelerek kapatmıştım… Bir de işte bu bloklarla ilgili şeyler teori dersinde hani biraz daha anlatılırsa hangi blok ne işe yarıyor diye öğrenirsek o faydalı olur. (T1_S3)*

Student T1_S10 has emphasized the topic she has had difficulty to be supported with more explanation in theoretical hour. She thought that harder topics such as lists could be explained more in theoretical hours.

> List things' function could be taught with more explanation… It could be explained in theoretical hour and implemented in lab hour. (T1_S10)
>
> *Listedeki şeylerin ne işe yaradığını daha açıklayıcı bir şekilde öğretilirse… Teorik derste anlatılsa şeyde de uygulaması yapılsa labda (T1_S10)*

Some students even needed verbal explanation for the parts that were written in the tutorials. That could show that demonstration and verbal explanation is needed, even if the information was presented in the tutorials. In the first implementation, the course was based on laboratory primarily, while theoretical hours were used for algorithms and recitation hours for homework. According to the feedback of the students, explanation, and features of each block (or code) could help students to understand the concept better. Based on the feedback of the students from the first semester, at the second semester of the course a recitation-lecture was added to the schedule, in which rather than making new activities, the instructor explained the example of the last week and the blocks used on that example in detail, and answered questions of the students. Since students demanded a detailed explanation about specific blocks, theoretical hours were changed to add explanations of the blocks used in last week's tutorials do, in detail.

Peer Support and Idea-Pitching

In addition to recitation hours, student T1_S2 also thought that it would be better to use the theoretical hours also as a discussion and idea-sharing place. According to him, students can explain their homework and the process of it as they did for their final project. Instead of the Facebook environment, he was thinking that sharing ideas face to face would be better to benefit from the ideas of each other.

> For theoretical hours, there could be homework like stuff. We could share the ideas like “I used this block because…” Just like, we made in our final presentations. The same thing could be in the theoretical hours. I used this because… Because different ideas will be shared. (T1_S2)
>
> *Teorik saatte ev ödevi tarzı şeyler olabilirdi. fikir paylaşır şu bloğu kullandım çünkü hani en son uygulama sunumunda yaptık ya bunları kullandım çünkü bunun için teorik derste de bunların üstünde konuşulabilirdi. Bunları kullandım çünkü ya da çünkü farklı fikirler gelecek. (T1_S2)*

One of the suggestions of the students was group project. The group project is commonly used in programming courses especially for the final project of the course. Student T1_S2 was thinking that coming up with a project idea was hard and providing project ideas and distributing those ideas among groups would have helped them to work more effectively.

> It became harder to come up with a project idea when you let us decide. I was not sure what to do, what to make. I mean providing options and based on difficulty distributing them between two people, three people, four people groups could give better outcomes in my opinion. (T1_S2)
>
> *Siz serbest bırakınca zor oldu projeyi seçmek. Ne yapsam ne etsem kararsız kaldım… Seçenekler sunup zorluğuna göre 2 kişi 3 kişi belki 4 kişi o tarz bir şeyler yapılıp daha iyi sonuçlar alınabilir bence. (T1_S2)*

Student T1_S10 suggested the group project for the final project since she thought that with a group they could have developed more complex applications.

> There could be a group project; I mean everyone could come up with an idea. We could have developed better things. (T1_S10)
>
> *Grup projesi olabilirdi daha hani herkesten fikir çıkardı. Daha güzel şeyler de geliştirebilirdik. (T1_S10)*

Student T1_S11 also suggested group project to improve their responsibility towards the mission and each other. She also suggested that having group work could help them to overcome their mistakes by helping each other and support each other to come up with a new idea.

> Group work could have improved our responsibility. Could be effective if there was something like… For example, we will develop an application. Everyone could have come up with an idea or could have thought what to make in that application. Since there was a shared commitment to it, when someone stuck, s/he could receive help from others. It could be more effective. (T1_S11)
>
> *Grup çalışması şöyle bir şey etkin olurdu belki biraz sorumluluğumuz artardı. Mesela bi uygulama geliştirecez herkes belli bi kısmında fikir yürütebilirdi ya da uygulamada neler yapılması gerektiğini düşünürdüortak bir amaç için olduğunda hani yapamadığı yerlerde de hani diğer öğrencilerden yardım aldığında daha etkin olabilirdi. (T1_S11)*

While group projects have some problems on its own, App Inventor environment also does not support the project works effectively since the developers could not work on one project simultaneously. Instead of developing one application together, students could be their support-buddies. Check their applications and help each other to overcome their mistakes. They could be responsible for his/her buddy's project up to an agreed on grade ratio. They could have regular meetings to check their final projects or homework.

4.5.2.4 Supportive Materials

Student T2_S6 thinks that there should be additional materials each includes the concept itself instead of an example. Students want to open those documents and check the attributes and how-to-use of that block when they have a difficulty. Attributes of each block were provided on MIT's website. However, it was in raw text and disconnected from its use. So additional support documents could be helpful for the student.

I mean, generally, it goes from example-wise, we need to scan the whole example and try to extract. However, at least, in some fundamental topics…I mean, if there were documents like "this is the logic behind this, and you can use it like that." It will be easier. (T2_S6)

Yani genelde hep örnekler üzerinden gitmiş bütün örneği tarayıp ordan onu çıkarmak. Ama en azından biraz daha belli başlı konularda hani değer alanlarını bilmiyorduk bizim örneklerimiz dışında. Hani şu şekilde yapılır bunun mantığı şudur gibi biraz daha dokümanlar olsa kolaylaşırdı. (T2_S6)

Student T1_S1 has found a supportive material which was prepared for making up for a holiday week as very helpful. He thinks that the explanation of each block would help their learning to be more effective.

It is so good to be taught with PowerPoint (slides). You put both the block and the picture of it. I think that should be increased. You made some activity with us in lecture, you can increase the number of those… Both could progress simultaneously. For example, if we could learn in theoretical hour and afterward in (lab) course, we can implement, it would be more effective, in my opinion. (T1_S1)

Powerpointlerde öğretilmesi çok iyi hem bloğu bloğun fotoğrafını da koymuştunuz siz derste görmüştük. Bence onlardan daha fazla olmalı. Onlarla birlikte bize de etkinlik yaptırmıştınız derslerde onları arttırabilirsiniz... Beraber gitse mesela teorik derste öğrenilip bir sonraki derste yapılabilecek şekilde olsa çok etkili olablir bence. (T1_S1)

4.5.3 Summary of the Dynamics, and Evaluation of the Course

In this part evaluation of the course including the challenging concepts, reasons of being challenging, and probable solution to teach them, suggestions and strategies to make an introductory programming course more effective will be emphasized in brief.

- Challenging Concepts: Same name different feature makes the concept more difficult, putting emphasis on then with basic examples and support material could help to solve the problem
    - Variables
        - Misconception: confusing with mathematical variables
        - Frequently changing the value of the variable makes it harder to understand the concept
        - Possible solutions: Emphasizing the difference between mathematical and programming variables, starting with basic examples focusing solely on variable use, underlining the role of the variables constantly
    - Clocks
        - Misconception: Confusing with the real clock with a fixed interval
        - Found difficult even by the students with programming experience since they did not use of clock in a textual programming course
    - Database
        - More emphasis on it needed
        - Useful basic examples should be used as a resource for students
- Course Dynamics and Suggestions
    - Need more Discovery Learning
        - Restructuring the tutorials to keep students as active thinkers during the course. After the first two weeks tutorials should have:
            - Less step by step explanation
            - Missing parts
            - Fewer images
            - Just textually explained steps
            - Added challenge
            - Dynamic feature to show the solution after some try or time

            - Develop more activities for faster learners/makers
        - The more practice, the better learning
    - Homework
        - Weekly supportive homework
        - Should involve problem-solving and discovery
        - Should not be too challenging for the first time learners
    - Using both Top-down and Bottom-Up Approach could help
        - deeper learning
        - students to understand the function of each block
        - keeping student mentally active
    - Theoretical Hour should
        - Have more verbal explanation of the function of the blocks
        - Be recitation for the lab hours
        - Encourage peer support and idea pitching
    - More Supportive materials should be provided for the students
        - To help them overcome the difficult concepts
        - Which could be more effective to remember the features of a specific block in brief

4.6 Examination of the Products and Course Progress

In addition to the qualitative data examined above, final projects and homework of the students could encapsulate essential information about the progress of the students and their engagement in programming. For the first semester, two homework were given to students, one with a walkthrough tutorial and one with a no tutorial. For their final project, students were asked to create an application with App Inventor. Students were free to create any application they want under the supervision of instructor without any boundaries. As expected, there was no difference among the first homework of the students, since they were asked to use a walkthrough tutorial. However, after the observations, it came up that using only walkthrough examples could lead to obstructing learning. Due to that observation, some in-class assignments were given without any tutorial. Similarly, the second example was given without any tutorial. For the first semester, the second homework and the final projects were investigated to see the progress of the students individually and based on their characteristics. Majority of the participants were from CEIT and EME programs. Those programs was not very close to each other in terms of university entrance exam score. CEIT students approximately need to be in first 50.000 students to be accepted to the programs. On the other hand, EME students approximately need to be in the first 12.000 students in the same exam. The minority of students were from ESE program (55.000), Physics program (70.000), and Business Administration program (4.000) as it was stated in the participants part. Additionally, it should be stated that Business adimistration program accepts students with a different type of exam score (Turkish-Mathematics), while all of the other programs accepts students from Mathematics-Science exam score. Nonetheless, the university exam score was not affect the success of the students. Each group has students with different success levels before and after taking the course.

Another point was knowledge level of students before and after taking the course. As it was stated before the participants were selected intentionally to be heterogeneous through the purposeful sampling method. Some students have the programming knowledge before taking the course, while others do not. Examination of the products put forward that pre-knowledge level of the students was not the project an important difference between their projects.

4.6.1 Examination of Products of the Students

4.6.1.1 First Semester

In the first example, it was expected of students to create an animation in which a car and countdown timer will appear on the screen. After the countdown timer reached to zero from ten, the car which is an imagesprite will start moving through the right side of the screen. To achieve this students are free to use any component unless it is logical. Four expected concepts to be used in this example were variable, clock (as a timed loop), if statement, and imagesprite. The homework students sent were examined one by one and by group characteristics.

Student T1_S1 who is a male Elementary Mathematics Education (EME) student used defined and used variable successfully. He also used the double Clock block which is a timed loop and need to be used two times in this example one for the countdown, other for the move of the car image. Additionally, if statement was also successfully implemented. The student was among the students who were experiencing misconception about defining and using the variables. Additionally, he also had some problem with using the clock concept, however, he used both of them effectively in this example. The probable reason behind the improvement is switching from walkthrough tutorials with image of each step to in-class assignments with no tutorials. After the problem solving during the class and demonstration after the students' solutions, students stated that learning with discovery is better.

A student with similar background and characteristics to T1_S1 was T1_S2 who is also a male EME student with variable misconception at the beginning. The difference of T1_S2's was that he used a label component as the value holder instead of a variable. The use was not wrong, however, it could be an indicator of negative attitude towards variables since this student was the one having problems with defining and changing the value of the variable. The application was working correctly, however, there was one logical error regarding the clock loop. Student used the same speed for the imagesprite, instead of increasing it. Other EME students, T1_S3, T1_S11 who are female students of the course were also successfully completed the homework, efficiently. The last male EME students T1_S9 has also completed the example successfully. Interesting difference about this student's homework, there were unnecessary code blocks, nevertheless, T1_S9 was the most successful student among the EME students. This phenomenon showed itself in other successful students too. For example, T1_S4 who was a CEIT student with previous programming background had also used unnecessary blocks in his homework. Similar examples were also seen for the successful students of next semester. This could be due to proactive inhibition for the students with previous programming experience. Code blocks of those students showed that the students who were more interested and more successful had tried to develop the application via alternative ways or with new blocks experimentally. For students who are willing to create a more advanced application should be encouraged, but providing a guide or a roadmap for them could help them not to lose in creating something novel. It will be mentioned in the discussion part as develop-it-more activities.

There was no significant waste of resources in the projects. Some characteristics of expert programmers listed as efficiently organized knowledge schemas, organize the knowledge according to underlying algorithm rather than syntax (Robins et al., 2003).

Nearly all of the students have effectively implemented the programming concepts and according to an algorithm rather than focusing on syntax. Regarding the complexity of the project, one CEIT student created a first-aid application which did not include any blocks that showed his programming skills. Additionally, one ESE student did not use variables as the application's value holder. She used label areas instead. It is important to make students feel free regarding choosing their projects. However, there should be some criteria about programming concepts to be integrated into the projects. In total, 7 students preferred creating application related to their career which was mostly educational games or applications. Remaining 4 students developed various applications. One female student (T1_S11) from EME department developed an interactive campus map. Another female student from Physics (T1_S10) department developed a grade calculator with note-taking feature. The interesting fact about the project of those two students are (1) personal need related, (2) included not mentioned concepts. Additionally, one male EME student (T1_S9) developed a game (breakout clone) as the final project. He also developed a package tracking application willingly in addition to his final project. One male CEIT student's (T1_S5) project was not appropriate in terms of its coding which was a first-aid application. As it can be seen in the Figure 4.12, the majority of student projects was educational applications or games. This finding was also coherent with the finding from interviews and observations in which relation to the career was one of the important instigators of product development as a part of the need of the student.

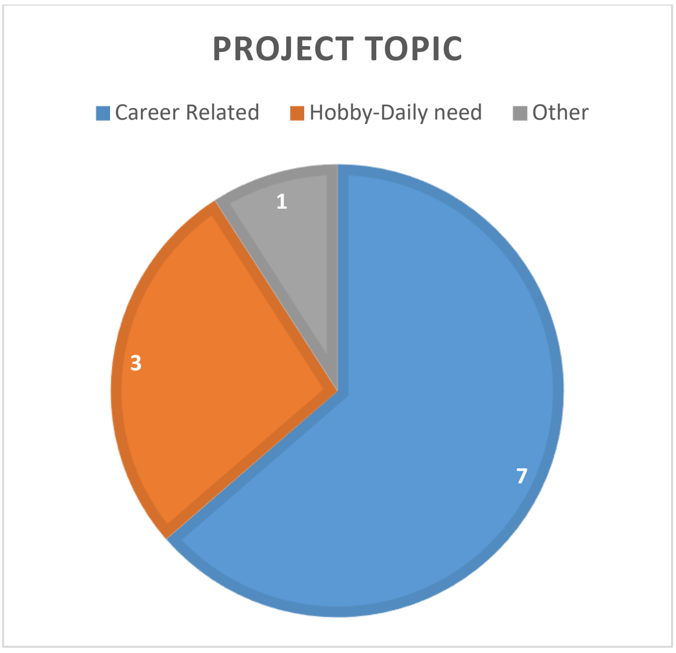


Figure 4.12 Categories of Project Topics (First Term)

Examination of the projects showed that 5 of the projects were very complex to create in which students used additional information that was not mentioned in the course, 4 of them has mediocre complexity (used all the mentioned concepts), and 2 of them were not very complex to create but useful (Figure 4.13). Analysis of the products showed that regular in and out of the classroom homework helped students to understand the more difficult concepts. Additionally, explaining blocks by using a top-down approach could also be useful for students.

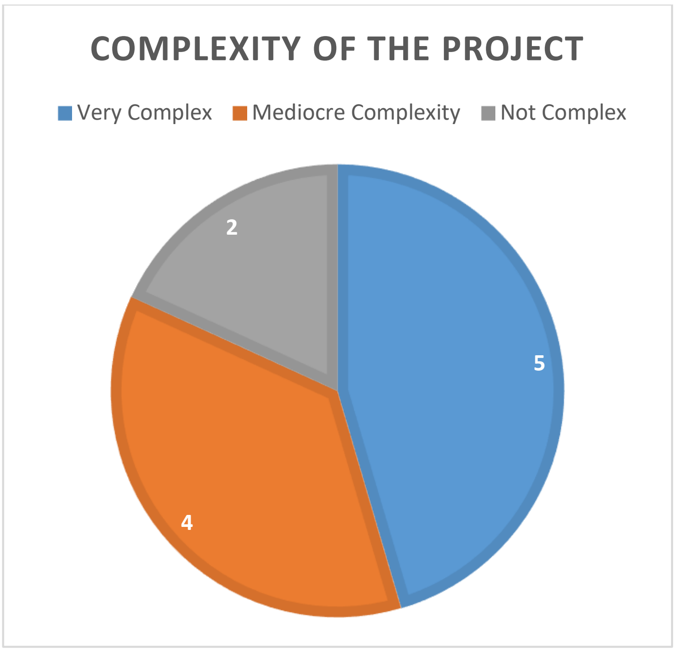


Figure 4.13 Complexity Level of Projects (First Term)

4.6.1.2 Second Semester

Participants of the second semester consisted of 7 students (4 males, 3 females). Five of the students were from CEIT department, 1 of them was from Business Administration department, and 1 of them from EME department. In the second semester, there were 5 homework, 1 project as the out of the class assignment of the students. First homework was with a step-by-step tutorial. Following 3 homework were with specific tasks to be completed. The last homework was creating a part of a professional game (Dumb ways to die). Students selected the part they want from the application.

Examination of all the homework showed that using homework is an effective intervention to make students revisit the programming concepts they have learned. It also motivated them to discover the new concepts of programming. A similar finding with the first semester is the better the students at programming, the more unnecessary blocks were used. Phenomenon probably appeared due to desire to explore and seek alternative solutions. This could be seen as a waste of time and resources, however, it can also provide a chance for students for further exploration. Homework of two students (both female, one from CEIT, one form EME) who were having difficulty understanding some concepts were similar to in-class examples. They probably used the tutorials as a guide for themselves. Providing additional information with tutorials could help students who are using tutorials as a resource. Creating dynamic tutorials could be one way as a solution which was also emerged from the interviews.

Examination of the second-semester projects in terms of their topics (figure 4.14 showed that 4 of 7 projects were games. From those 4 students, 2 students were from CEIT, 1 from Business Administration, and 1 from EME department. 2 CEIT students have developed an educational game. Remaining 1 CEIT students developed an application for calculating the calories of the foods. In total 6 of 7 students have developed a game, the probable reason behind this could be that most of the tutorials and homework were game or game related examples. Building the course around a theme like games, education, or commercial applications could help students to be more engaged.

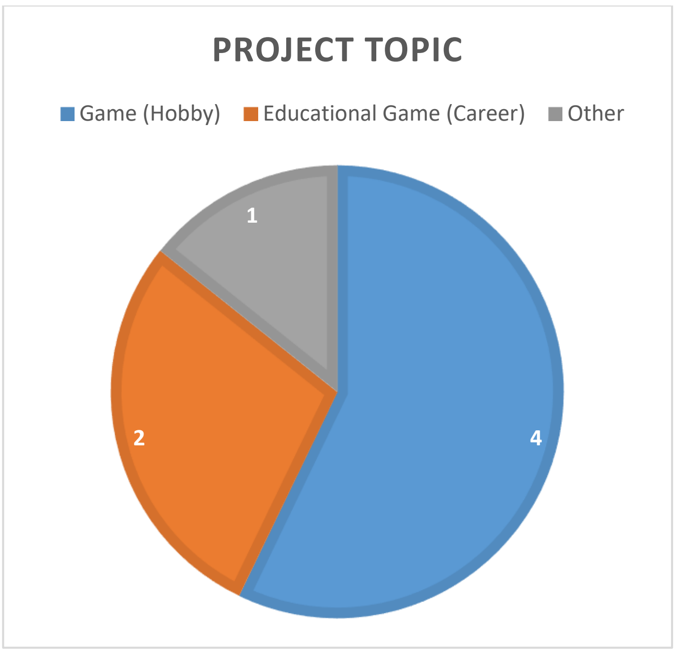


Figure 4.14 Categories of Project Topics (Second Term)

Regarding their complexity, analysis of the projects put forward that that 4 of the 7 projects were very complex (figure 4.15). They involved all of the concepts that students learned throughout the course. Remaining 3 projects had mediocre complexity.

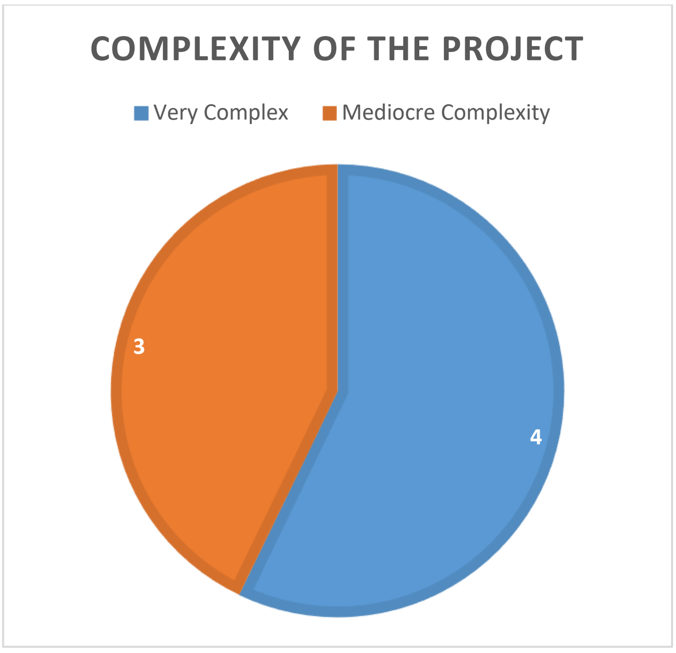


Figure 4.15 Complexity Level of Projects (Second Term)

4.6.2 Progress of the Students based on Characteristics

For tracking the progress of the students the first semester were used since the major changes were made in the first semester. Elementary Mathematics Education students from the first semester (3 males, 2 females) were compared since they are the largest group with similar characteristics without any programming knowledge before this course. The compared group CEIT students (3 males) was not very homogeneous in terms of their programming knowledge, however, they all are at least taken one programming course before. Based on three phases about their programming knowledge was compared to see if there were any significant intervention. Observation notes and analysis of in-class assignments, homework, and projects through rubrics showed that in-class implementations without any step-by-step tutorials were showed the real knowledge of students. However, by providing more examples and homework with variables helped students to understand the variable concept and use it effectively in their homework and project.

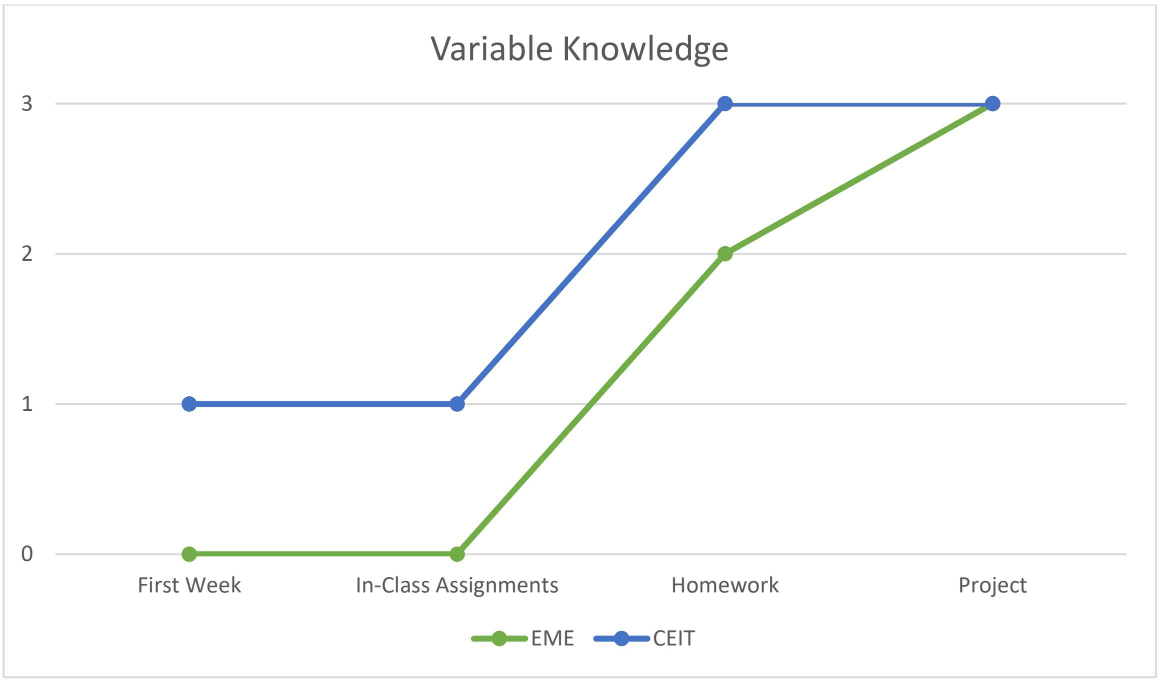


Figure 4.16 Variable Knowledge of Students based on their Departments

When projects of the students were compared based on their gender (6 male, 5 female) regarding the first semester, there was no significant difference in terms of the knowledge of programming concepts.

4.6.3 Interventions throughout the course

As some of the interventions for the problems mentioned before, some of the strategies and recommendations emerged during the course based on the problems or observations of the instrucr listed in table 4.1 below. At the start of the course, instead of a theoretical example a tutorial with visual product at the end was provided to students. Students were amazed and motivated when they see a working product at the end of the tutorial in their phone. This observation led instructor to put emphasis on products of the tutorials for each week. Interviews confirmed that focusing on the products during the course hours was found effective in terms of motivation and effectiveness. Rather than implementing the simple algorithms, intergrating the same algorithms into the products is found more effective. This interventention named as the product-first strategy. In this strategy, it is recommended that putting emphasis on designing the in-class tutorials to have a useful, working, and purposive product at the end. Achieving engagement of the students through product-first approach could also increase the effectiveness of the course. Additionally, it was also found that using games as the products has additional positive effect for the engagement of the students. After the first semester, it was decided that using games as the course theme would help students to understand the programming concepts better, since it would build on top of a similar context.

In addition to using the game theme, during the course hours it was observed that using spiral approach in which students learn key concepts simple to complex with a repetitive nature. Concepts offered in previous weeks should be re-offered to students with additional features and information. In accordance with using a theme strategy, using spiral approach also helped students to understand the complex topics like variables, loops, arrays etc.

Table 4.1 *Interventions based on problems or observations*

Problem-Observation	Intervention
Engaging effect of visual, useful, working, purposive products	Focusing on Products (Product-First)
Seeking help by messaging (shyness)	Encouraging communication in and out of the classroom
Following in-class tutorials without thinking	Providing in-class assignments, develop-it-more activities
Having difficulties understanding the variables, clocks etc.	Specialized small examples for the concepts + repetitive examples with spiral approach
Engaging and meaningful use of game examples	Using games as the course theme
Positive effect of recitation and discussion	Support-Buddies and Discussion environment

Throughout the course, a Facebook group was setup as the communication medium to enable the constant and open communication. However, during the course some students asked each other students through the direct messages. Since this would affect the openness of the course, instructor intervened by encouraging the open communication by asking simple questions and giving positive reinforcements to simple questions. Another problem observed throughout the course was following the in-class tutorials without thinking. In this problem, students as they also reported during the interviews, copied the tutorials without understanding the functionality of the example. To overcome this problem, instructor provided in-class assignments each week without any tutorials which were based on last week's information. Students had difficulty seriously to complete the assignment. However, in the following weeks they learned to use resources to solve the problem. In addition to in-class assignments, more challenging tutorial desing was also considered. In the second semester, develop-it-more activities were provided to students who finished the tutorial steps in which students need to enhance the application with new features offered at the end of each tutorial.

Another observation was the need for face-to-face discussion and group work/support which was demanded by the students during the interviews. Using a group project was not feasible due to the nature of App Inventor Environment. Additionally, using group work could lead to unbalanced workload among students, especially in an introductory programming course. To overcome this problem, students recommended to have a recitation hour where they could ask question for the topics they did not understand and discuss about their project. A discussion and recitation hour was set up which was similar to studio approach. Additionally, it was decided to have support-buddies approach in future in which students were assigned to help each other's project. It was not applied in this study completely, however, through the communication medium students were assigned to help each other's problem.

4.7 Overall Summary of Findings

Interviews with students and observations throughout both semesters formed the main data of this dissertation. Quotations and notes from this data sources gathered around 5 different themes. Those themes are (1) Communication, (2) Contributions of the course, (3) Motivation, (4) Programming and programming environment, and (5) Dynamics and evaluation of the course.

One of the themes emerged from the interviews with students and observations throughout the course was communication. It was not expected to emerge as a main theme, however, it is found that communication is one of the most essential factors that influence the motivation towards a novel subject, if not the most essential. It was put forward that it is important for all courses, but in difficult topics, students needed more in and out of the classroom communication. Communication is not only essential per se, but it is also a source of scaffolding and information. Communication between students and with instructor was one of the most mentioned elements that influence the motivation towards the course and success of the students according to themselves. Feeling comfortable to ask questions any time they needed help making them more relaxed for the times they had stuck. In addition to out of the classroom communication, in-class communication between students was also found helpful to help them to overcome their mistakes. Communication and collaboration of students should be encouraged when they are making the in-class examples. As it was mentioned before out of the classroom also should be encouraged. One of the efficient ways to do that was using a mutual communication medium. Choosing a medium that commonly used and providing both synchronous and asynchronous communication environment. Facebook was chosen by the students. Being instantly notified by the application of Facebook, providing multi-directional, interactive, and open communication was the most mentioned positive features that Facebook environment have. Facebook also provided a resource hub to students since all of the tutorials, resources, and homework including their feedbacks were shared on the Facebook group. Students also stated that they used the group both for communication and for the resources it possessed. The medium has some negative aspects as it had positive aspects. Even though the instructor constantly encouraged students to ask any question on the Facebook group, open to everyone, some of the students used direct messaging to ask their questions to instructor and their friends because they were ashamed of their “to easy to ask” questions. Constant encouragement solved that problem, however, instructor should be aware of that problem exist in the digital medium as well as in the classroom. Additionally, choosing and using one medium could make some students who are not using that medium as much to feel left out.

Another theme emerged from the qualitative data was contributions of the course. Under this theme, positive sides of the course that was reported by the students, the concepts they have learned consciously and unconsciously, the signs of learning and motivation towards the programming concepts were gathered. As a sign of learning and embracing the topic, future plans could be seen as an important indicator. Even the students not related to computer education reported that they could teach programming and its logic in their professional lives. Some of the students stated that they could develop their own application for career-related or as a hobby. A group of students was also planning to use App Inventor to support the other courses they will take. The course was effective in terms of relating to career and future of the students. Being able to develop their own applications made their knowledge more relatable to their everyday life and careerwise. As it will be mentioned further in the following theme, motivation, being able to develop their own application might be the most important factors that influence the motivation towards the programming according to interviews with students. It helped students to connect their knowledge with their world out of the classroom. Therefore, it kept them motivated towards programming and prevent them from cutting their interest. In addition to future plans of the students, they reported that they realized with algorithmic thinking style their perspective towards their daily lives were changed. They started to think differently as they stated. They realized "computational thinking" -even though they named it differently- the thinking style they were using throughout the course was also similar to the one they used their everyday lives. By realizing they were using algorithms, they consciously started to use that thinking style. Students could be more involved in the course if the knowledge from the course was connected to their daily lives and provides a value for today or for their future. Emphasizing and linking the programming knowledge such as computational thinking could also help them in their professional lives.

The third theme was motivation. Understanding what motivates students towards the course could help the instructors to lower the dropout rate as well as increasing the success rate. According to interviews and observations, students find it motivating to work in a visual environment. Both the developing environment and the end-product were visual in this course. Students reported that developing a visual product and doing that in a visual environment also motivated them towards the course. Moreover getting visual feedbacks and cues from the environment made it easier for them to understand and overcome their mistakes which leads to an easier and concrete learning environment. Designing the course step by step and constructing on top of the previous week's examples also motivated them towards the course and programming. A non-intimidating design could lower the drop-out rate dramatically, especially for an introductory programming course. As it was mentioned before, one of the most mentioned motivators about the course was creating a working and useful product. Instead of algorithm focused examples, algorithm-embedded useful examples should be used to keep students motivated towards what they are learning and what they are developing. Seeing a useful end-product after each tutorial was the strongest motivator for this course. When the students with previous programming experience compared the course with their previous courses, the mostly mentioned difference was the end-product of what they have learned. They defined the examples from their previous courses as meaningless algorithm tutorials. Students also reported that the sense of learning something new every week was one of the factors kept them motivated to come to the classroom each week. Other than providing new programming concept, providing a new component with different feature could also help the instructor to motivate the students towards the course. Students also need to have a practice every week instead of theoretical information similar to the previous finding.

Another important aspect to be examined was the programming and programming environment. This theme has emerged from the interview questions and examination of instructor throughout the course. As it was mentioned before App Inventor, a visual programming environment was used as the programming environment of this course. The comparison of the environment to textual environments could provide important inside for programming instructors. According to interviews with students, visual environment helped them to overcome their fear towards programming. They thought that programming was hard before the course. Students reported that seeing that environment allowed them to program just by drag and dropping helped them to build self-confidence about programming. When they compare visual and textual environments and offered to select one of them, most of the students selected a visual-only environment. They explained their choice based on three main reasons: no-syntax problems, immediate feedback and a user-friendly environment, and concrete products (as it was mentioned in the motivation theme). Students reported that visual environment provided them scaffolding, and preventing them to make mistakes by providing visual cues. However, there were some students reported that textual programming should be integrated into App Inventor environment to "edit the code" would make the environment more effective and flexible. Similarly, flexibility and adaptability were seen as the only advantages of the textual programming environments. Some students stated that visual-only environment would not be flexible enough for advanced programming. Hence, it could be beneficial to use a visual programming environment with a support of a textual environment for introductory programming courses. When students were asked to evaluate the App Inventor. Similar answers emerged. Additionally, students reported that App Inventor had some technical deficiencies such as crashes and slowing, and lacked some features like undo, offline working and visual flexibility. Evaluation of environment showed that a mature visual programming environment with textual programming support could be the ideal environment for an introductory programming course.

The fifth and last theme was dynamics and evaluation of the course. Under this theme, challenging concepts, suggestions of students about the course, and possible changes were gathered. The programming concepts students have had difficulty with was asked to students. Three different concepts come forward: variables, clock, and database. The most common mistake was with variables due to the confusing the concept with mathematical variables. The clock which is basically a time-based loop was also found hard to understand by some of the students. Those two concepts were emphasized by using them in multiple examples which helped students to understand. On the other hand, the database concept was not focused enough since it was one of the last examples. Breaking down the challenging concepts into smaller examples could help students to understand the concept from multiple aspects. In addition to challenging concepts, what students would add or remove from the course was also investigated. According to interviews, one of the suggestions come forward that tutorials should not be well-structured. Tutorials should have fewer images and could be removed after a couple of weeks. This was not a consensus as it was expected. Therefore, a dynamic tutorial could be provided to students as students have problems, more helping images would be revealed. Moreover, some students asked for even more practice and homework. They stated that more practice in and out of the classroom would help students to learn programming even better. Some students also suggested (without knowing the concept) that using both top-down and bottom-up approach could be beneficial for students. Providing finished examples, and examples with missing pieces could also be used as well as bottom-up examples. Students from the first semester also suggested a recitation hour to further explanation for the examples of the week. This suggestion was used at the second semester and found useful by the students. Additionally, students offered to make group projects, however, both the environment would not allow an effective group work, and since it was an introductory course some members could have left behind. Rather than that students could be assigned to support each other for ideas and help.

CHAPTER 5

DISCUSSION AND CONCLUSION

An introductory programming course is difficult for many students, even the ones considered as digital natives (Kafai & Burke, 2017; Wiedenbeck, 2005). Learners' success in the future regarding computing and programming will depend on the introductory course they took (Grover, Pea, & Cooper, 2015). Programming has a central role in computing and computer science especially regarding the implementation of the concepts (Rountree et al., 2013). Programming is seen as a difficult skill to learn and acquire and research has shown it needs special attention to teach (Mannila, Peltomäki, & Salakoski, 2006; Sajaniemi, Ben-Ari, Byckling, Gerdt, & Kulikova, 2006). In classical programming teaching, a general purpose language is taught, and the course is more concerned with the concepts the language have rather than programming concepts (Orfanakis & Papadakis, 2014). Even more, than 30 years before this study, introductory programming courses were growing in K-12 and higher education, and faculty spends a significant amount of time on course development and instruction since (Anderson & Skwarecki, 1986). Some common problems in teaching and learning programming listed as (a) motivational problems, (b) lack of agreement upon efficiently teaching concepts, and skills in introductory courses, (c) methodological problems in teaching programming (Buitrago Flórez et al., 2017, p. 841).

Research question of this study is "What are the instructional strategies and recommendations to develop an efficient, effective and engaging introductory programming course for non-CS majors?" Efficient, effective, and engagement aspects of the research study are simply based on their dictionary definitions. Efficient is defined by Merriam-Webster Dictionary as "capable of producing desired results with little or no waste (as of time or materials)" ("Efficient," n.d.). Hence, being efficient for an introductory course as in this study means that being capable of providing essential concepts of the programming to students in one semester or less time. Effective is defined by Merriam-Webster dictionary (“Effective,” n.d.) as “producing a decided, decisive, or desired effect” which is equipping students with the capabilities of developing their own application at the end of the course, understanding the essential programming concepts. Lastly, engaging defined by Merriam-Webster dictionary as “tending to draw favorable attention or interest” (“Engaging,” n.d.). In this study engaging used as a synonym of motivation and keeping the desire to learn programming and continuing the course.

This study focuses on extracting instructional strategies and recommendations for an introductory programming course. Four themes emerged from the qualitative analysis of the course: (1) Communication, (2) Contributions of the course, (3) Motivation, (4) Programming and programming environment, (5) Dynamics and Evaluation of the course. Themes and codes were investigated in-detail in the results part. On the other hand, discussion part converted the results into applicable, and ready to implement instructional strategies and recommendations with the support of literature. It was aimed to help other instructors to design their courses through the strategies and recommendations extracted from interviews and observations from two separate but connected courses with the literature. It should be noted that this study does not focus on what happens inside of novice programmer’s brain. It rather focuses on to understand what could be done to stop them from being afraid of the course, improve their self-confidence, and help them learn programming in a one-semester course by providing strategies and recommendations to instructors. Some of the strategies and recommendations offered in this study are not solely for instructors, but also for the learners. Robins et al. (2003) suggested that strategies should be more explicit and should be discussed as a part of the lecture in introductory programming courses. The common mistake about teaching programming is focusing on syntax rather than pragmatics of writing a program (Sajaniemi et al., 2006). Soloway (1986) also suggested that rather than teaching syntax, strategies should be explicit for the learners. Strategies and recommendations from the findings and literature will be explained and will be presented as a final model in this part of the study.

5.1 Preparation for the Course

Before starting the course there are some points that the instructor/teacher should take into consideration. In this part of the study, the preparations for the course is investigated. Three crucial points come forward among others: (1) Knowing the learners, (2) Choosing the programming environment, (3) Choosing the communication medium. Learners and their characteristics should be the ones who steer the wheel from the first fork on the road. As the focus group of this study is non-CS major university students, all decisions, strategies, and recommendations are based on such learners. Some of the strategies could be relevant to other groups as well. Selecting, changing, or manipulating the strategies are up to the instructor. Choosing the programming environment should also be based on learners and the interest of the time. “The process of teaching and learning not only involve learners but a set of situations in which teachers stage knowledge about programming.” (Rogalski & Samurçay, 1990, p. 158). Before starting the course it is essential for an instructor to be competent about which the programming language or environment he will teach and to whom he will teach. Another important aspect of this study is communication. Establishing the communication is crucial for any kind of course. In subjects that found difficult by the students, it is more important. Using communication tools and media is one of the factors that influence and enhance communication among the students and between students and instructor. Three points mentioned will be examined further in the following sections of this study.

5.1.1 Knowing the Learners

The most important factor in any kind of learning environment is the learner itself. As it is suggested in any instructional-design implications, one of the most important elements of an instructional design is to know the audience or the learners, if not the most important (Reigeluth, 1999b). “Curriculum designers and teachers need to consider student profile…” (Araujo et al., 2018, p. 1). The main distinction in programming based on the experience level of the learners: novice and expert programmers. According to Winslow (1996), it takes 10 years to turn a novice programmer into an expert one as a general knowledge from the studies. Winslow (1996) also reported the characteristics of novice programmers as having limited superficial knowledge, lacking an adequate model, using general problem-solving strategies rather than specified one, using line-by-line, bottom-up approach. The instructor should take those characteristics of the learners into consideration before starting to design the course. This course and the study focused on novice programmers rather than expert ones.

In this study, an initial survey was conducted to learn the characteristics of the learners. As it was mentioned in the participants of the study part, students are not from the same programs and have same knowledge level. Hence, the course was designed accordingly.

Prensky (2001) defines the new generation as ‘digital natives’, however, being a digital native does not necessarily mean that the new generation has all the skills they need. One of the main differences between expert and novice programmers is that experts retrieve the plan from memory, while novices must create plans from scratch (Robins et al., 2003). Experts programmers and novice programmers have different characteristics as it was mentioned. Having learners from different backgorund needs a more flexible design. Strategies like develop-it-more activities, and support-buddies which will be mentioned later depended on heterogeneous foundation of the course. The instructional strategies offered in this study could be commonly used for introductory courses, however, it is not intended to be used for computer science majors. Defining the characteristics of the audience should be the first step before starting to design a course. Learning the general and specific characteristics of your audience regarding their programming knowledge level is an important indicator for the instructional design. While the all of the participants of this study were novice or non-programmers, coming from different background was also another factor that affected the design of instruction. Similarly, the perspectives of learners out of the computer programming field (e.g. non-CS majors) are also different. While CS majors have similar target to reach the at the end of the course, non-CS majors could have different purposes since they have a different background. Their purpose of taking the course should be learned and analyzed to begin developing the progress of the course.

Rather than trying to transform their perspective, instructors need to keep them in the field and keep their motivation and self-confidence high. Results of this study showed that even the students from unrelated departments considered a career in programming after their self-confidence levels rose during the course. Following strategies and recommendation for the course, design was created according to students from the range of no knowledge of programming to with programming experience of fewer than 4 years. Instructional strategies and recommendations could be modified or manipulated according to the characteristics and the development of the students through out the course. Knowing what works and what does not work for the learners, would be clarified as the instructor know the learners. Using the Design Based Research features including reshaping the course based on the interative cycles, and being flexible throughout the course was one of the strengths of this study. It is recommended for the future courses as well.

5.1.2 Choosing the Programming Environment

Programming and programming environment was one of the themes emerged from the interviews. Therefore, it is an important aspect for this study to choose programming environment. Programming environments are systems used by programmers to develop and test programs which provide a range of functionalities and used by both novices and experts (Deek & McHugh, 1998, p. 133). Teachers/Instructors are the ones who decide the programming environment and how to use them in their courses (Levy & Ben-ari, 2009). Curriculum revision in programming education through the years changed the programming languages and environments, however, paradigm stayed the same (Armoni & Gal-Ezer, 2014). Integrating a new programming environment is not separate from other design strategies and the educational paradigm. Foundations of this study also based on the experiences of the researcher and the development of a new programming environment which has a potential for the first-time learners. The programming environment is crucial for programming education, however, programming education should not be confused with programming environment education; on the contrary, the focus point should be the pedagogy and the strategies (Grover et al., 2015). Wing (2008) also emphasized that it is essential not to confuse the learning about the environment with learning programming concepts. Even though, the focus is not the programming environment, it is one of the most important aspects that could affect the learning the essential concepts. This study tried the combine the positive aspects of a programming environment with the instructional strategies developed during the progress of the course. Therefore, investigating the recent programming environments is an essential recommendations for the instructors. Based on the results of this study, one of the important aspect was the features that programming environment offered. It is critical the choose relevant and effective programming environment or environments to have an effective, efficient, and engaging course. App Inventor was chosen in this study. Positive aspects of the environment were explained to give the readers an insight about the effect of the environment and which features could help students to learn better and more easily. The negative aspects were also reported in the results part, however, since the focus is choosing the relevant environment, the aspects that environment need to have were reported in this part of the study.

The programming environment comes with challenges as well as the opportunities. The programming interface could intimidate new learners (Wyeld & Barbuto, 2014). Normally, learners get to know the environment and language, learn and become comfortable with the environment, thereafter students start to build programs (Romeike, 2008). Although the programming environment is not the main focus, students have to learn one and it plays an important role in the learning process (Mannila et al., 2006, p. 211). Overcoming the mistakes and creating error-free programs is a challenge and a struggle for non-majors who are taking introductory programming courses (Wiedenbeck, 2005). Choosing a relevant programming environment could shorten the steps to build an error-free program. Starting with a relevant programming environment as important as the other instructional strategies, since it shapes the perspective of students towards programming and it impacts student/teacher dynamics (Deek & McHugh, 1998; Felleisen, Findler, Flatt, & Krishnamurthi, 2004). This study used the App Inventor environment which let students start creating their own visual and working applications from the first week, as it was one of the most engaging and motivating interventions different than a traditional programming course. During the writing of this study mobile applications were very popular among the society regarding both daily lives and profession to be mastered. The popularity should be another aspect to be considered before choosing the environment.

Environments that support computational thinking could learners to focus on abstraction, automation, analysis by minimizing the effort spent on the coding process (Repenning, Basawapatna, & Escherle, 2017). Such environments “…must be consistent with the activities of actual learning situation and support the entire problem solving and program development process" (Deek & McHugh, 1998, p. 172). Bruce (2005) suggested using a different programming language or environment than the ones in advanced courses could help students to take the first step to the programming. As generations are changing and technology is developing, it is not possible to say learning strategies and tools should stay the same for students. Many existing environments are too complex since they are designed for professionals which overwhelm beginners (Kölling, Quig, Patterson, & Rosenberg, 2003). The course designed are not for professional, however, it is expected to be a step for non-programmers to continue on advance programming. Therefore, the selected environment should be flexible to satisfy the learners with different intention to take the course. Technological developments allowed new environments to be developed to support learning and problem solving (Deek & McHugh, 1998). Using textual environments solely for the introductory programming course could lead to disengaging students with boring examples. Choosing a visual programming environment instead that offers essential programming concepts could be more relevant for introductory programming courses. "Visual programming environments enable the learner to write programs using graphical notation." (Ben-Ari, 2013, p. 53). Results of this study showed that having a visual environment to program and having a visual product at the end are both effective, efficient, and engaging for the students. Choosing an appropriate programming environment could help reduce extraneous cognitive load, and by freeing of cognitive resources from aspects of language and syntax, it could be channelled into concepts and procedures (R. Mason & Cooper, 2013). Instructors do not see syntax errors as severe problems, however, they could easily frustrate the students especially in the introductory programming courses (Mannila et al., 2006). As the literature suggested, the common problems with programming environments having syntax problems and errors which could cause a great extraneous cognitive load. The features (or lack of feature for some environments) are in direct relation with motivation of the learners. Using a visual programming environment could also be a solution for reducing the cognitive load and motivate the students toward the topic.

There should be some criteria to choose a programming environment regarding an introductory programming course for non-CS majors. Many instructors do not use an integrated environment, because they could not find a suitable one which causes loss of opportunities, time, and efficiency (Kölling et al., 2003). Most of the courses programming languages like C++ and Java, which are both popular in the industry, however, their suitability in introductory programming education is questionable (Mannila et al., 2006). Moreover, traditional programming environments provide less support for problem-solving (Deek & McHugh, 1998). Visual programming could be a solution to support problem-solving. Grover et al. (2015) stated that visual programming courses are relatively easy to use, and to avoid errors of programming syntax for beginner level learners which let them focus on designing and creating. Visual programming use in this course also supported the focus of creating rather than focusing on the syntax.

The main criticism against using easier languages or environments is the chance of running into the wall when students have to switch on to a more complex one (Mannila et al., 2006). However, firstly, switching is not a must for non-majors, and secondly, it is a risk that could be taken instead of losing the students' interest in the first course. Connecting the textual programming with visual programming is also another strategy emerged from this study which will be mentioned later. Using an only visual programming environment is not a problem, but having the benefits of multiple environments could be beneficial for students with different programming background. Another risk is the difference between the knowledge level of students. While first-time learners could create their programs without knowing the language syntax, students with previous knowledge of textual programming could find visual languages confusing (Deek & McHugh, 1998). Using textual programming environments to support the visual environment could also be beneficial for students with previous experience with textual programming environments.

Even though visual programming environments have a colorful interface, it could still provide sophisticated features as the textual environments do (Ben-Ari, 2013). As it was mentioned before, App Inventor environment was chosen for this course. App Inventor is used to create applications for Android devices as it was mentioned before. App Inventor is a free to use environment, the majority of K-12 teachers polled that as the language they most wanted to learn (Margulieux, Catrambone, & Guzdial, 2016). Studies show that students are well equipped with the latest mobile technology (Iqbal, Chowdhury, & Harsh, 2013). App Inventor was an effective and engaging environment according to the opinions of the students, and observations of the instructor. App Inventor distinguishes from other programming environments by gathering some characteristics in it: (1) visual environment with visual products, (2) allow to develop mobile apps, (3) provides immediate feedback and live testing, (4) suitable for adults and children, (4) allows extensions and modules (Arduino, Lego Mindstorms etc.). However, for the future courses, there could be a better or more relevant programming environment for the target group. Even though there could be limitless possibilities, some criteria are emerged and required for introductory programming courses. Results of this study showed that some features came up among others.

New learners of programming could be shocked on their first encounter with the environment to deal with the different types of difficulties (du Boulay, 1989). The programming environment should be easy and simple to use, to develop, and to learn since it is an introductory programming course. Especially the students with no programming knowledge emphasized the importance of having an easy to develop environment in their hands. They connected the being easy with being visual which is not irrational since seeing the code as blocks and producing a visual product was what motivated them in the first place. Being visual is not a must, however, this study showed that it has a big potential regarding both learning and motivation. Deek and McHugh (1998) suggested that the focus of the programming courses should be problem-solving activities rather than emphasizing on syntax. According to Robins et al. (2003), typical introductory programming courses are syntax focused and knowledge-driven. However, programming is "beyond mere command of the syntax and semantics of a programming language" (Deek & McHugh, 1998, p. 130). A visual environment could help to shift the focus to using strategies by removing the overemphasis on the syntax. Additionally, visual or textual, the environment should provide constant help to students with cues and scaffolding. Without immediate help and guidance, students could recede from programming. Finally, as one of the important findings of this study, the product is very important for the student. Winslow (1996) reported that even the students who understood the syntax and solved the problem have trouble creating the program by combining the feature they learned into a working program. An easy and helpful environment could let students focus on problem-solving and product development which is also a rewarding outcome for the students. The technical and product-wise capability of the environment should also be taken into consideration since some of the students criticized the lack of features in App Inventor. The programming environment should provide immediate, consistent, detailed and informative feedback to the learner (Robins et al., 2003, p. 158). The main criteria based on the findings of this data that could be used before selecting the environment are summarized in figure 5.1 below.

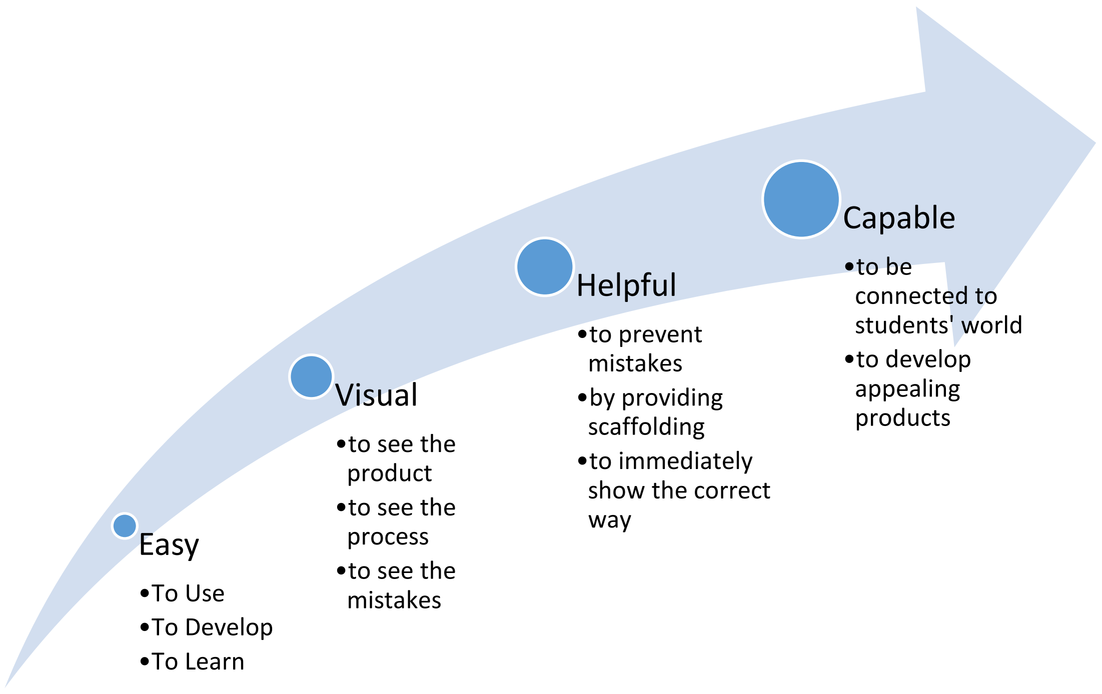


Figure 5.1 The Criteria to Select the Environment

Students in this study compared the App Inventor as a visual programming environment and the textual programming environments they have used before. The comparison table below has come forward (Table 5.1). The comparison revealed that that visual programming environments are more relevant to start learning programming. However, in some point students would need to switch, if they are planning a career in programming.

Table 5.1 *Comparison of Visual and Textual Programming Environments*

Visual	**Textual**
Practical	Theoretical
Simple	Complex
Product-Focused	Algorithm-Focused
Limited	Flexible
Less Likely to have User Errors	More Likely to have Syntax Errors
More Likely to have Software Errors	Less Likely to have Software Errors
Immediate Feedback and Testing	Feedback after Compiling

Based on the data analyzed in this study, using a visual environment is more relevant for an introductory programming course. App Inventor was one of the most suitable ones based on the needs of the students and the instructor. However, visual programming environments could limit students, especially if they are seeking a career in the computer programming field. As it can be seen in Table 5.1, the two negative opinions about the visual programming were to have errors and its limited capability. Errors could be fixed in future updates of the App Inventor, however, it is less likely to use the visual programming as an environment for an expert programmer. Therefore, students with programming career in mind could have negative opinions towards visual programming, since visual environments could lack a way of expressing the solution similar to traditional programming environments have (Cambranes, 2013). Some of the advanced functionality of textual environment is not needed in the first-year learning in programming, therefore it could be traded off for an easier environment (Kölling et al., 2003). This study also showed that students who have taken a textual course before thought that the examples in textual programming courses are pointless. Even without a visual programming environment, students sought to create visual and purposive products rather than sole algorithms (Robins et al., 2003). Each environment have the advantages of its own (Noone & Mooney, 2017). On the other hand, the strategy to overcame disadvantages of both environments could be creating a blended environment regarding the visual and textual programming as it was suggested above. Starting the course with visual programming environment at the center, supporting and explaining the examples with textual examples could help students to see the visual programming as a step to advanced programming. In another saying, for one or two semesters long visual programming course could be used as training wheels for the students with future plans about programming, and easy to learn programming knowledge for other students. Even though the students with no programming knowledge preferred visual only environment, the instructor should not cut the connection between visual and textual programming environment (Homer & Noble, 2017). Moving from visual programming to textual programming is difficult for the learners (Kaurel, 2016). Using both environments could easen the transition switching to a text-based environment from a visual one, as Araujo et al. (2018) reported in their study. However, it is not studied to see if the students would switch to a textual environment.

A disadvantage that visual environments potentially hold is that focus could shift into dragging and dropping (by trial and error) rather than designing and constructing the program's structure (Ben-Ari, 2013). It should be noted that the use of visual and easier environment is for the first time learning, not to master the use of the environment as a career choice, and students who seek a career in programming should be forced to explore professional environments (Kölling et al., 2003). Supporting the course with a textual programming knowledge is important to help students to feel comfortable about switching to a more advanced environment.

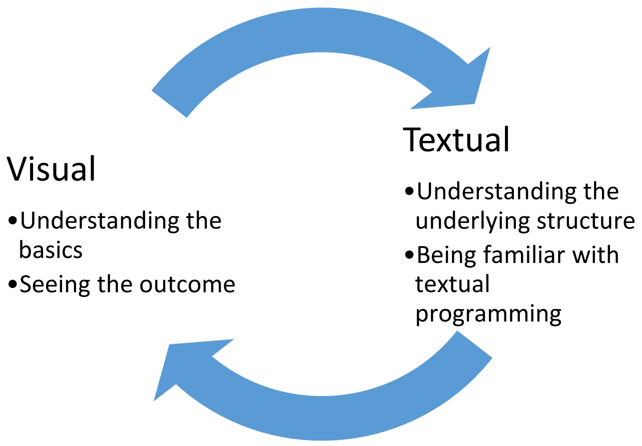

Visual
•Understanding the basics
•Seeing the outcome
Textual
•Understanding the underlying structure
•Being familiar with textual programming

Figure 5.2 Benefits of using textual programming to support visual programming

Kalas and Winczer (2008) suggested that three stages of working with computers which are (1) using it; (2) understanding it; (3) creating with it. This study suggests that learners should understand it by using while creating with the environment, rather than having sequential stages. Programming environment could ease the three stages as the App Inventor environment did in this study. Providing a relevant environment with an environment capable of creating visual products facilitate the learning by doing.

5.1.3 Choosing the Communication Medium

One of the results that the researcher did not predict was the effect of communication medium on engagement level of student, and effectiveness on the course. Using a communication medium to enhance communication could be useful for students who need help out of the class hours. Giving in-person teaching support and necessary feedback to students requires a significant amount of time commitment (Ichinco, Zemach, & Kelleher, 2013). Using a communication medium is helpful for instructors but it is their duty to use it to enhance the learning experience through the communication medium (Manasijević et al., 2016). According to Wellman, Haase, Witte, and Hampton (2001), use of the Internet could supplement the face to face interaction and extend the time spent on communication. Communication was found one of the most important factors that influence both engagement of the students, and effectiveness of the course. To ensure motivation and success while designing an engaging introductory course, communication of instructor with students and communication among students are crucial. To support the communication among the students and with instructor, an online communication medium was planned to be selected. In this study, communication medium was decided through a survey conducted with the students. Among options, all of the students participated in the study chose Facebook as the communication medium. A Facebook group was set up for the course according to the demand of the students. Social networking sites have the potential to provide easy access to discussions, instant communication with other students (Maleko, Hamilton, & D'Souza, 2012). Facebook is commonly used for meeting new people and communicating with friends (Ellison, Steinfield, & Lampe, 2007). Educators also considered Facebook as an alternative learning management system or educational tool to support learning and activities (Dennen & Burner, 2017). In Charlton, Devlin, and Drummon's (2009) study Facebook was also the student's choice as the communication medium that meets their needs which would support the communication and work collaboration. Using Facebook for education could increase communication, collaboration, and material/resource sharing (Manasijević et al., 2016). All three aspects were observed to increase in this study. Among the adults aged 18-29, Facebook is very popular which makes it a relevant choice for education to benefit from its popularity (Dennen & Burner, 2017). Participants of this study also were 100% Facebook users; only 1 of 18 participants was not a daily active user of Facebook. He was using Facebook for educational purposes only. As the results of this study put forward, students found Facebook as the communication medium helpful. Rather than having focused on a sole medium, it is important to examine the positive and negative sides, and the views of the students towards the medium to determine the medium of the courses.

One aspect to care about is that the communication medium should be popular and used commonly among students before the course. According to Ellison et al. (2007), 94% of their participants who are undergraduate students were already Facebook members. As of 2018, Facebook has 1.47 billion daily active users around the world (Statista, 2018b), and 44 million users in Turkey (Statista, 2018a). Results showed that it is important for students to use the medium regularly. For many students, Facebook is part of their daily routine, and they defined it as part of their everyday activity (Ellison et al., 2007). Additionally, students stated that being able to easily access Facebook group through mobile phones anytime and anywhere was also essential for them. Being able to access Facebook anywhere makes learning of programming easier (Maleko et al., 2012). Another feature that students stated as a positive aspect is being notified instantly. Tracing the updates and being notified are also important for students, according to the interviews. While tracing the updates in sites like wikis, LMSs, and forums may frustrate students, media like social media, e-mail etc. could easily notify students about the updates (Thomas, King, & Minocha, 2009).

Another feature that came up in the interviews as an advantage is being interactive, allowing multi-directional and open communication. Facebook is an environment that users could present themselves in their profile, they can post comments on each other's pages, join virtual groups of interests, and they can learn about and from each other (Ellison et al., 2007). Facebook groups provide environments for interactions to be used anytime anywhere, that encourage engagement with programming and enhance the learning of programming (Maleko, Hamilton, D'Souza, & Scholer, 2014). Comparing to e-mail, students in this study found Facebook more interactive, positively informal, and instantaneous. Similar results also came forward in Charlton et al.'s (2009) study, students found easier, more active, and more efficient to use Facebook instead of using e-mail. Students also compared the Facebook group with the learning management system of the university in favor of the Facebook group, similar to Maleko et al.'s study (2012). Instant notifications and instant feedback increase learning and interaction (Maleko et al., 2012). Discussions and seeing each other's posts should be encouraged since they are beneficial for every student, even if they did not face that specific problem (Dennen & Burner, 2017). Students in this study stated that they benefited from the posts and discussions of each other since every question and answer provides a new perspective for them.

Another positive aspect of the communication medium that students used throughout the course is using the Facebook group as a resource hub, repository of examples. "Novices were more successful when they continued referencing an example throughout a programming task" (Ichinco & Louis, 2016, p. 261). Using the older examples, assignments, and the projects of other students helped the learners throughout their projects when they faced a problem.

One problem throughout the study was direct messaging among students to seek help from each other. Similarly, in Dennen and Burner's (2017) study, some students preferred to use private communication. Some of the students in this study also performed a similar behavior. The possible reason behind this is the fear of being embarrassed. Performance-avoid goal orientation in which students' purpose for doing well is to avoid being embarrassed, or looking stupid (Cheong et al., 2004). The instructor should encourage the all of the students to ask questions open to everyone in communication medium. When students with task goal orientation which is learning for learning sake (Cheong et al., 2004) shared their question, this would help the others to have the courage to ask questions and seek for help open to everyone. First-time learners could show dependency syndrome in which students depend on others to complete their assessment (Bati et al., 2014) which could lead to direct messaging as this study put forward. More transparent communication could fix this problem, or remove the syndrome stepwise as the students see each other 's progress and questions with their answers. Facebook group could also help shy students to initiate communication and encourage the students asking questions to and responding to questions of others by lowering the barriers to participation (Ellison et al., 2007). Some students may not feel as comfortable as the other using Facebook or any communication medium. Especially, if the medium is one of the environments they use for their personal life like Facebook (Dennen & Burner, 2017). Those students could be informed to open a new account for the course since the main communication would be in Facebook group.

Using a communication medium as a resource hub to constantly support the students. In addition to tutorials, support documents should be created and provided for the students through the communication medium. Students in this study stated that they extracted the needed information from the tutorials. It would be more helpful for novice programmers to reach the support documents at any time they needed. Especially when they encountered with a novel situation, they would need support documents more. Support documents similar to a cheat-sheet could help them not to lose themselves in such a situation.

To sum up, a communication medium, not necessarily a Facebook group, that (1) is popular and commonly used among students; (2) allows multidirectional, open, and interactive communication among students and with instructor; (3) could be used as a resource hub; (4) notifies students instantly and accessed anywhere/anytime they need; could enhance learning as well as communication.

5.2 Implementing the course

5.2.1 Product-First Approach

Romeike (2008) stated that the problems presented at the introductory programming courses are math problems instead of programming problems. Even though the math is in touch with programming, this could affect the motivation of the students negatively. Rather than accurately coding a program or algorithm, nowadays students can create authentic applications such as playable games, sophisticated animations, or digital stories as signifiers of success (Kafai & Burke, 2017). Accurate to purpose of this study which is developing a course that is engaging, effective, and efficient, creating such applications as examples could effect engagement which is directly in relation with effectiveness. Students should be constructing programs that matter to them rather than building meaningless algorithms (Lye & Koh, 2014). Gross and Kelleher (2010) suggested connecting the code to observable output since it would help students to interpret the programming concepts according to its functionality. Results of this study also showed that products or artifacts are one of the most important factors that influence the effectiveness and engaging nature of programming instruction according to students. Students also thought that after creating something on their own, programming is “cool” rather than “scary”. Seneviratne (2017) stated that programming courses often focus on theory rather than application. Rist (1995) also emphasizes the importance of breaking out of theory and stepping into the action by creating a concrete level of design (p.537). Kurkovsky (2013) stated that many CS students disappointed due to the irrelevant examples which are boring or non-related to the real-world applications which sometimes lead to a change of program for the students. Romeike (2008) stated that students performed and motivated better if the task is meaningful and the product is relevant to their reality. Relevant to the Merrill’s (2013) First Principles of Instruction’s fourth and fifth principles (Learning is promoted when new knowledge is applied by the learner, and learning is promoted when new knowledge is integrated into the learner’s world), knowledge and learning are directly related with the products that students created. Experimentation and learning-by-doing are important in programming education (Christiansen, 2004). Similarly, Anderson (2015) stated that “doing” is what makes learners shifting from abstract information to knowledge of practice. In this study, students learn programming by creating the applications each week with the help of tutorials. Soh et al. (2007) suggested having hands-on practices on curriculum. Additionally, homework and final projects of the students were also products that were relevant to the learners’ world. As the literature suggested, the programming education is shifting from math problems, and sole algorithms to real-world applications and meaningful products. Interviews and observations supported this idea. Especially, the students with previous programming experience compared the previous courses they take with the one in this study. The main positive difference they stated was difference between the features and characteristics of the products they produced.

Difficult and meaningless tasks could lead to loss of interest. (Howles, 2009). As it was mentioned in the findings part, students found the creating a useful, working, and purposive product was seen as one of the most engaging parts of the course. Robins et al. (2003) also stated that "the reinforcement and encouragement derived from creating a working program can be very powerful" (p. 158). In the light of the findings, it is possible to say that in addition to the student-centered design, putting the end-product in the center according to the characteristics of the students could influence the motivation of the students. Lye and Koh (2014) also stated that creating something concrete was one of the popular interventions for students. Instructors should focus on the design of tutorials and artifacts that students are to create. Instructors should help students to see classes as task goals rather than performance goals "…in assignments and activities, minimizing competitive structures, and adopting evaluation practices consistent with task goal orientation." (Cheong et al., 2004, p. 16). Results of this study put forward that the end-product should be/have (a) visual/concrete, (b) purposive/functional, (c) related to real-life, (d) integrated algorithms, (e) relevant to students where it is possible (figure 5.3).

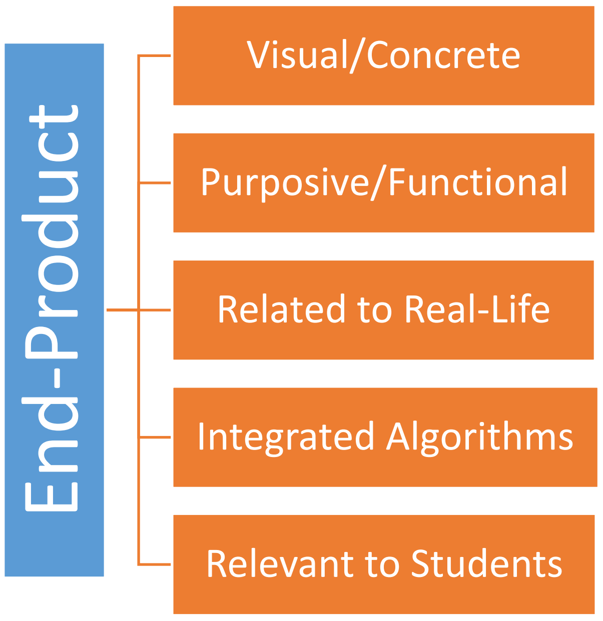


Figure 5.3 Characteristics of end-product

While Merrill (2013) stated that "learning is promoted when learners apply their newly acquired skills" (p. 25). In this context, learners are getting the skills and knowledge while they are applying it. Davies (1993) separated the two level of programming,

programming knowledge (the declaring how a programming concept works) and programming strategies (the way knowledge is used and applied). Using a product-first approach could allow students stepping from programming knowledge to programming strategies from the beginning of the learning. According to First Principles of Instruction, knowledge is presented in different levels. In product-first approach, information should be hidden in application part (figure 5.4). By implementing and adopting the knowledge while creating an application take the level 2 to the same level with level 0. This study put forward that demonstrating and explaining the examples after the implementation help students understand the concepts better. After the implementation and understanding the concepts thoroughly, providing problem-centered acitivies to students with in-class assignments could be beneficial. Effectively implementing the skills students learned could be performed by solving a problem rather than following specific steps (Merrill, 2013). Putting the activities at the center while designing the course could enhance the effectiveness of the asssignments and examples, problem-solving capabilities of the students could be enhanced. As the effectiveness of the products increase, effectiveness of the course will also be increased due to the focus on implementation part.

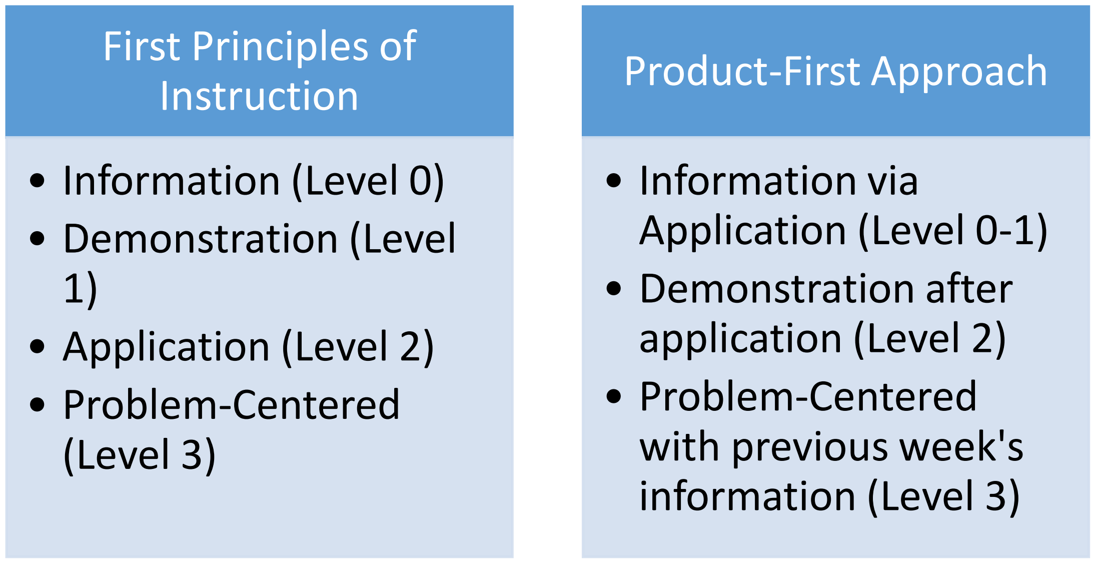

First Principles of Instruction
Information (Level 0)
Demonstration (Level 1)
Application (Level 2)
Problem-Centered (Level 3)
Product-First Approach
Information via Application (Level 0-1)
Demonstration after application (Level 2)
Problem-Centered with previous week's information (Level 3)

Figure 5.4 Levels of Information in Product-First Approach

5.2.1.1 Visual and Purposeful Products

As the previous part suggested, focusing on the design of the examples are crucial to motivate students as much as increasing the effectiveness of the course. Students put forward that it is essential to have visual products at the end of each week and example. The first example of a traditional programming course is writing “Hello World” to the screen by using code. Guzdial and Soloway (2002) suggested that instead of textual outputs like “Hello World” examples, students of the new generation demand more visual, and multimedia programs to develop. “Working on easily accessible tasks, especially programs with graphical and animated output, can be stimulating and motivating for students.” (Robins et al., 2003, p. 158). In this study, students developed a soundbox application in which there is a button with a image when it is clicked it plays a sound. Installing and using the app they developed had a very dramatic effect on students. It made them excited and motivated to develop more. Interviews put forward that developing visual, working, concrete, and purposeful product was what motivated students most with similar finding emerged from the observations. The more engaged students are, the more effetive course was.

Using drag-drop programming by developing animations and visual products are found to reduce syntax errors, reduce the frustration of students and introduce the programming in a fun and engaging way (AlHumoud, Al-Khalifa, Al-Razgan, & Alfaries, 2014). Studies show that games could be used as educational game tools since they increase the motivation level of the students (Chao, 2006). Developing games as the artifacts might have the similar impact for many students. Developing games and editing the code making changes visible and concrete which makes novice programmers more comfortable in the coding environment (Wyeld & Barbuto, 2014). As it was stated before, visual products also have the effect of increasing engagement towards programming. Interviews put forward that motivation of testing their own games motivated students more than playing commercial games. Observation notes also showed that playing their own games had a positive effect on the motivation of students. There are few studies that report about using mobile games instead of computer games to increase the engagement and interest level of learners (Kurkovsky, 2013). Teaching is more beneficial when the instruction is “...interesting, meaningful lessons that inspire students’ interest and focus attention on the task itself, as well as to minimize the emphasis on grade competition” (Cheong et al., 2004, p. 14). Guzdial and Soloway (2002) suggested focusing on the multimedia products that students would want to produce to teach programming is a crucial strategy to enhance the engagement level of the students. “Students become intrinsically motivated when learning tasks give them senses of competence, autonomy, relatedness, and purpose" (Herman, 2012, p. 371). Moreover, Bergin and Reilly (2005) put forward that the more important, useful, and interesting the tasks are the better the programming performance. One of the clear findings of this study is products that are relevant to learners' interest increase the engegement of the students and effectiveness of the course according to the experience acquired from this course. Seneviratne (2017) stated that developing a working game, in the end, to play made students excited about their product. As similar studies suggested using games and purposeful applications had a similar effect on novice programmers. Kafai and Burke (2017) also stated that designing an application increases the interest and success rate of the learners and sharing their artifacts attracts them to programming. When designing the examples, assignments, and homework for the course, it should be taken into consideration products like games and animations emerged at the end are seen as an important source of motivation for the students. It is a relevant strategy for the introductory programming course students. Developing games as the product was inverstigated further in the following part of this study.

5.2.2 Spiral Approach with a Course Theme

Product-first approach offered in the previous part focused on creating examples and assignments that have a visual/purposeful product at the end to increase the engagement of the students and effectiveness of the course. On the other hand, providing method of the examples is also crucial for an introductory course. Winslow (1996) stated that "Good pedagogy requires the instructor to keep initial facts, models, and rules simple, and only expand and refine them as the student gains experience." (p. 21). The spiral approach was chosen to reinforce and strengthen the newly acquired knowledge of students in a recursive manner. "The spiral approach is the parallel acquisition of syntactic and semantic knowledge in a sequence which provokes student interest by using meaningful examples, builds on previous knowledge, is in harmony with the student's cognitive skills, provides reinforcement of recently acquired material and develops confidence through successful accomplishment of increasingly difficult tasks." (Shneiderman, 1977, p. 193). Normally, spiral learning's focus is curriculum, however, it can also be adapted to separate course by presenting contents repeardly and with gradual increase in complexity. (Araujo et al., 2018, p. 2).

Learning needs to build on existing mental models of them. Using frequent hands-on activities which consists of tasks of increasing difficulty is promising more success than infrequent large assignments (Wiedenbeck, 2005). As the figure 5.5 visualized, spiral approach in learning let learners to introduced with concept, thereafter the same concept was presented constantly in more complex examples with new concepts introduced over time. Through the successive cycles, the basic concepts introduced, further details are presented over time by expanding the details to expand the knowledge level of the students (Jaime et al., 2016). Increasing complexity and difficulty would help students reach mastery without having any difficulty. In this study, concepts were presented through the tutorials first. However, observations suggested that using basic examples in the start is more beneficial for increasing the effectiveness of the learners and the course. Spriral approach improved the learning performance, motivation and quality in some studies which could lower the dropout rate and it is a coherent strategy for all educational levels (Araujo et al., 2018; Jaime et al., 2016; Jing, Cheng, Wang, & Zhou, 2011; Veladat & Mohammadi, 2011). Spiral approach is an effective methodology to teach programming for first-time learners, even for 10/11-year-old learners (Yovcheva, 2008). Students in this study are higher education students, they also found the flow and implementation of spiral approach beneficial. In this study, students stated that they feel more organized since the provided examples were connected with each other. Observations also confirmed the finding as the student used the previous week’s concepts and examples to solve the current problem. Similartly, sudents in Araujo et al.’s study (2018) found the spiral approach as organized and helpful, which facilitated their learning.

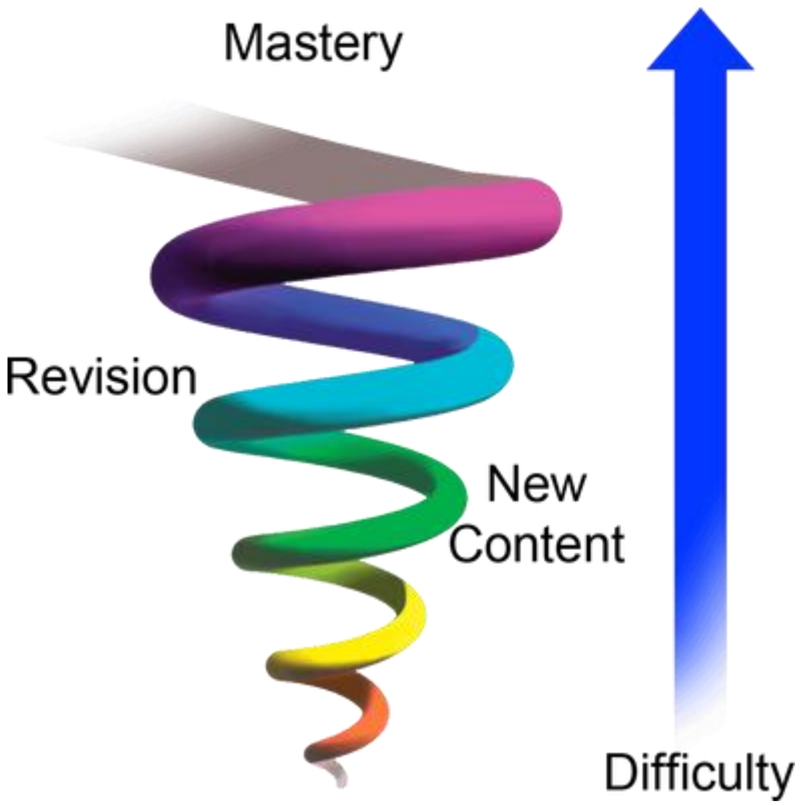


Figure 5.5 Spiral Learning - Novice to Master (Writer, 2017)

This study embraced the spiral approach from the beginning since it was appropriate for introductory level learning and it was relevant to Merrill’s First Principles of Instruction. Merrill (2013) suggested a four level instructional strategy starting with information-only, continue with information and demonstration, plus application, and problem-centered information, demonstration, and application. Merrill’s strategies are also builduing on top of the previous strategy. Product-first approach manipulated the steps as it was presented in the previous part. Nonetheless, spiral approach similarly offered a way with increasing complexity through the examples. Spiral approach is also beneficial to reduce the extraneous cognitive load of new learners (Araujo et al., 2018). Implementing the spiral approach along with product-first approach, keep the students motivated and focused on learning new concepts.

According to students, keeping the sense of learning something new each week was one of the motivating factors for students to come to the class because they would like to learn the new features by creating their own applications. Another important motivator for the learners was developing something that they did not create before the course. It is essential to keep that sense throughout the course. Spiral approach is helpful to new learners since it repeatedly returns and connects the old knowledge with the new one at different stages of learning. Similar to spiral approach, constructivist theory also suggests that "…knowledge construction is recursive; therefore students will continuously build on what they already know…" (Bernard & Bachu, 2015, p. 285).

In addition to being compatible with introductory programming courses and product-first approach, some advantages relevant to this study of spiral approach for software development eduation listed by Spicer (1983) as (1) a natural, effective, and proven programming education technique, (2) useful for integrating both academic and industrial needs, (3) raises interest and self-awareness of the students, (4) provides hands-on experience, (5) increase self-confidence. On the other hand, there are disadvantages of spiral approach as well. Spicer (1983) also stated the disadvantages of the spiral approach, and the relevant one to this study is that it could be hard to provide feedback to students in large classes which is a similar disadvantage for other approaches. Spiral approach could also decrease the efficiency, in terms of providing time on a single concept. However, through product-first approach, programming concepts would not be solely focused on a single concept, which would not effect the efficiency of the course, since the course will end in estimated time and effort.

A very common motivational technique is to build a course around a theme (Merrill, 2013). Using a theme could help students the focus on the programming concepts while building on top of their experience each week rather than having disconnected examples. It is also compatible with the spiral approach since the examples would be congurent with each other. Students stated that they have the feeling of constantly learning something new on top of their knowledge. This course mainly focused on games which was found engaging and effective by the students. Use of games to teach programming explored in some studies (Araujo et al., 2018).While using games is not a must, its motivational aspect, relevancy to the spiral approach, and being compatible with the product-first approach make it a proper them choice for the course.

Game examples used in this course were open to development on top of the previous week. Since games play an important role for social connection among people, it could be a good medium to deliver the computational thinking, critical thinking, and problem-solving skills (Croff, 2017). Games are already a popular choice and an engaging way to teach programming in introductory courses (Al-Bow et al., 2009). Creating similar games to commercial games they play affect the motivation of the students, and their will to spend time and effort on developing one (Good & Howland, 2017). However, it should be noted by learners and teachers that an expert player in video games does not necessarily mean an expert in programming (Buss & Gamboa, 2017). Nonetheless “Mobile applications and game development may help students to better relate to the course material and make stronger connections to real-world applications and gadgets they see and use every day” (Kurkovsky, 2013, p. 138). Games for mobile devices are simple enough to create during the week and provide a chance for implementing programming concepts into a working application (Kurkovsky, 2013). Through spiral approach, games were getting more complex in each example of the course. As an example used in the course a basic animation using multiple images were used to introduce loop concept in programming. Students were excited about creating a game character animation. In the following week, a similar animation was used to implement movement of a imagesprite. The sense of continuing the development motivated students and put emphasis on the loop concept. Students without knowing the name of the spiral approach stated that constantly learning something new in addition to the examples from previous week helped them learn the concepts better.

Majority of studies show that students preferred developing computer games which were the main reason for many students to start CS program (Kurkovsky, 2013). Students in this also preferred developing games as the main examples of the course. Games are engaging when playing and developing which create a fun-flow experience (Chao, 2006). The part instructors should give extra attention to is that while using the motivational aspect of game-development, they should not leave behind the educational aspect (Kurkovsky, 2013). Using games could promote teaching and learning computational skills, and motivates students towards learning especially in introductory programming courses which are critical for students’ perspective towards programming (Kurkovsky, 2013; Quaye & Dasuki, 2017). Digital games are proposed to be used in introductory programming courses since games are attractive and motivational in nature (Quaye & Dasuki, 2017). The motivational side of the using games was realized by the instructor at the middle of the first semester which led instructor focusing on games more. The second semester was created based more on the game examples. Using games was effective in terms of motivation and learning.

It is important to note that game development(or whatever theme selected for the course) should not be the focus and instructors should not be afraid of wandering off from the theme. Some concepts may not be compatible with the theme and instructors could show the examples non-relevant to the theme. Using the theme in spiral approach is a strategy to increases the engagement (increasing the motivation level), efficiency (faster embracement of a new concept), and effectiveness (implementing the programming concepts in a familiar program). The theme should be flexible since it may not be compatible with all of the programming concepts.

5.2.3 Blending top-down and bottom-up approach

The two dimensions of programming understanding could be defined as comprehension which is describing how a piece of code works and generation which is generating the code that solves a problem (de Raadt, 2007). Both dimensions are important for the novice programmers. Traditional introductory programming classes tend to take place as traditional lectures which is compatible with top-down approach through demonstration, and presentation of finished examples, even though the hands-on practices in laboratory-based environments are more successful (Falkner & Falkner, 2012). While top-down and bottom-up strategies appear as two separate approaches, each has different benefits for learners and none of them should be left behind. Even though the spiral approach is more close to the bottom-up approach and it could have more potential for novice learners, top-down approach could be supportive for in-depth learning. Widowski and Eyferth (1986) suggested that employing a top-down approach to read conventional programs, and bottom-up approach for simple but unusual programs (cited in Robins et al., 2003). Gross and Kelleher (2010) also suggested for novice programmers to use and debug poorly constructed code to see unfamiliar codes for themselves and enhance their experience with different coding styles. In this study, it was observed that students acquired programming knowledge (which is required for first-time learners) by applying step by step instructions. However, for developing programming strategies, and applying the knowledge into novel situations, interviews and observations showed that using top-down approach could have the potential of providing useful information for first-time learners.

Students in this study also suggested using top-down approach as an alternative strategy. Rogalski and Samurçay (1990) reported that top-down approaches could be difficult for beginner programmers, therefore top-down strategy could be implemented by providing manipulation tasks to students to a finished application which students are not unfamiliar of. Robins et al. (2003) suggest using a design strategy relevant to bottom-up approach which starts with a cue, a direction, a level, and a type of link to explore further. Top-down approach was not implemented thoroughly during the course, however, an example was provided to students in the second semester. Additionally, they were using each other's projects to get help and review the code. During the interviews, students found reviewing each other's example beneficial.

Another benefit of using both approach is learning programming from different aspects. According to studies, writing a program and the ability to read a program have little connection, therefore it is beneficial to teach both of them (Robins et al., 2003; Winslow, 1996). Lister et al. (2004) also stated that one of the knowledge students are missing at the end of their study was the ability to read the code which even affected their problem-solving skills. Kafai and Burke (2017) also stated that using the top-down approach occasionally would give students the opportunity of seeing, modifying and building far more complex applications than they could build from scratch. Additionally, deciding what to add or remove from a built application offers a different experience to students which would be useful for building more complex applications (Kafai & Burke, 2017). This study showed that using visual programming and tutorials to build applications are helful but not sufficient in some aspects including lack of encounter with complex examples, debugging the errors of another application, and seeing other programmers' perspectives. All of which are expected requirements for a programmer. The examples could be prepared and be provided by the instructor or by assigning students to evaluate each other's assignment.

5.2.4 Dynamic Tutorials and Develop-it-more activities

Learners need more guidance and explanation for the topics they learned for the first time (Merrill, 2013). Structured laboratory assignments and handouts have advantages such as promoting cognitive abilities in comprehension and application (Soh et al., 2007). According to this information, this course was also designed as a well-structured and continued with detailed explanation and demonstration of each step.

Tutorials and assignments encourage and motivate students towards the new topics and completion of the next assignments (Willman et al., 2015). However, interviews put forward that some students needed more flexible and less demonstrating, and less structured tutorials. In this course, tutorials started with step by step well-structured tutorials with screenshots of each step. Students realized that after some time, they do not need that much structure and images. Providing problems with textual descriptions could be helpful for the problem-solving skills of the students (Hooshyar et al., 2014), which were provided as in-class assignments in this study. Providing unstructured tasks is also one of the demands that novice programmers have (Langrich & Schulze, 2015). Solely focusing on building a specific product could cause students to engage only on the environment rather than constructing a solution. Mannila et al. (2006) divided the problem-solving process into 4 stages: (1) understanding the problem, (2) coming up with a solution, (3) developing an algorithm, and (4) implementing the algorithm. Using well-structured tutorials might affect the problem-solving process negatively in which instead of coming up with a solution, students could wait for the step-by-step tutorial. Falkner and Falkner (2012) suggested providing gradually decreasing scaffolding while the knowledge of students is increasing. Higher motivation levels could be achieved by providing tasks that offer opportunities to be successful, but also yield challenges (Pintrich, 2003). In addition to examples, the programming environment is also essential to keep the confidence of students in higher state as it was stated before. Through the easy examples and an environment that digitally support and provide scaffolding to the students when they made mistakes would help students to feel competent about the programming. Providing only easy examples could bore the students who are good at programming. By using challenges, motivation of the students would be kept and they would stay in the flow zone (Csikszentmihalyi, 1991). While results of Soh et al.'s (2007) study showed that motivation and self-efficacy levels of students kept decreasing throughout the course. It is essential to keep the motivation high and constant throughout the course.

According to Csikszentmihalyi (1991), learners need to stay in the flow to be engaged in an activity. To achieve this goal, the task should not be too difficult or too easy. For a classroom with different knowledge level, it cannot be achieved with a static tutorial/task system. A gradually increasing difficulty/challenge level could be helpful for learners to stay in the flow. In the light of this information, it is suggested that tutorials and tasks should be open to discovery learning. Results of the study showed that it is better to provide well-structured, step-by-step for 2-3 weeks. After students got familiar with the environment and the concepts of programming, instead of step-by-step tutorials, a tutorial with fewer images, less direction, and missing steps could be offered. Top-down examples could also be helpful at this point. Rather than a step by step walkthrough, new tutorials should be a guide for students to solve the problems presented (see figure 5.9 below).

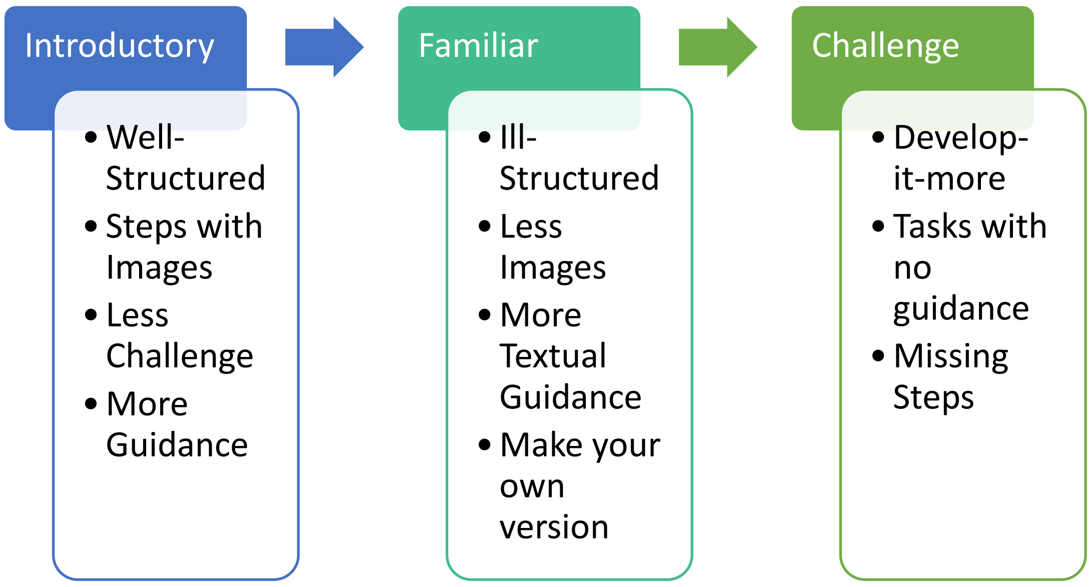


Figure 5.6 Progress and Steps Tutorials

Lye and Koh's (2014) study showed that the most popular intervention was providing scaffolding to students while they are constructing their own programs. Scaffolds could help them without copying the content in the tutorial. Additionally, based on the knowledge level of students, tutorials could be dynamic. One easy way to achieve dynamic tasks is to providing additional tasks at the end of each tutorial for the students who completed faster than their classmates. By providing 2-3 tasks to improve the application further, students would stay on the flow rather than simply waiting for their friends to complete the tutorial. For the first task, a "make your own version task" could be provided in which students should manipulate the application into a different one visually and functionally. In addition to that task, an open-ended task named "develop-it-more" activity could be presented to the students in which they could discover new features to complete the task. By providing open-ended tasks students could discover new features related to that specific example. Another way is by developing a webpage to provide tutorials based on the speed and success of the students. Tutorials could reveal the information if a student could not complete the task in given time. Develop-it-more activities could also be provided through a webpage.

5.2.5 Support-Buddies and Idea-Pitching

Aside from the other strategies and the programming environment, effectiveness, engagement, and efficiency of this course heavily relied on to communication. Constant and open communication between students and with instructor are found as one of the factors what increased the success and motivation of the students. Some novice programmers learn to program with little assistance, others learn to program with a great deal of assistance if they do not fail (de Raadt, 2007, p. 202). It is observed that students communicated and interacted with each other throughout the course. Similarly, Quaye and Dasuki (2017) reported that students constantly interacted with each other while trying to complete their program. Student-student interactions lead to a positive effect on student success (Maleko et al., 2014). Therefore, collaboration and communication should be encouraged in programming education. Students could learn teamwork, communication and sharing ideas to solve large and complex problems. (Kim et al., 2015). Throughout the course in-class and out-of-class communication and collaboration were encouraged.

Coding also is no longer an isolated activity, rather it is a social activity shared with other participants (Kafai & Burke, 2017). Most of the students demanded help from their peers or from the instructor throughout the course. Soh et al. (2007) proposed to encourage teamwork, and cooperation as a design strategy. Collaborative learning's major achievements are listed as motivation, social cohesion, cognitive development, and cognitive elaboration (Bernard & Bachu, 2015). Similar findings were emerged in this study. Students stated that communicating and collaborating with each other were seen as positively different feature of this course. Students listed communication and collaboration as one of the motivator for the course. Importance of collaboration regarding pogramming is emphasized by more than one study. As DeMarco and Lister

(2013) reported that 70% of their time, software developers work with one or more people, while they work alone in 30% of their time.

One popular suggestion came from students were formıng project groups. MacGregor (1988) stated that students who are working in groups tend to resolve problems with their peers rather than the instructor (cited in Williams et al., 2002). Project groups is a popular strategy to prepare students for group work in real life. However, there is a downside of group work when it comes to the classroom level. In most projects, the workload would not be shared equally. Sometimes students with better knowledge level took the responsibility and the other students just do the labor. Students who are better at programming work more while the others just tag along with them without doing much programming work. Especially in introductory programming, using group project approach could make students who are good at programming better and students who are bad at programming worse.

Additionally, environments like App Inventor does not allow simultaneous working on the same project as some other programming environment. This also restricts the group work specifically for this course. An alternative to project group and the most commonly used strategy that involves collaboration is pair programming strategy (Bernard & Bachu, 2015). In pair programming, two programmers use one computer collaboratively to develop a software, one person role as the driver and write/create the code, while others watch and direct him towards the solution (Hanks et al., 2011; Lewis, 2011). There is evidence that pair programming increases the competence, retention level, the success of students (Hanks et al., 2011; Lewis, 2011), however, for introductory programming, this also could lead to getting out of the flow especially for the students who are not using the computer. As Lewis (2011) also stated that students worked on their own computer were faster to complete examples on their own, which could be an indicator of pair programming could be unproductive for learning, engagement, and success of the new learners. Additionally, similar to group project, pair programming also could lead to distribution of unbalanced workloads, such as more successful student could do the most of the work (Mentz, van der Walt, & Goosen, 2008).

Collaborating to solve a problem should be one of the emphasized skills in programming education. Encouraging novice programmers to collaborate can help them to overcome some of the challenges they face, and by engaging discussion, they can see different viewpoints, alternative solutions to their problems, as well as offering their own (Bernard & Bachu, 2015). Teacher/Instructor could not possibly respond to all questions or requests for help, therefore the peer help is more beneficial, and valuable especially in crowded courses (Cheong et al., 2004). More knowledgeable students could help their peers to understand programming concepts that they could not understand on their own (Maleko et al., 2014). Courses could embrace a different collaboration approach based on their audience and the environment they chose. The studio-based approach could be one solution to enhance collaboration in introductory programming courses (Narayanan, Hundhausen, Hendrix, & Crosby, 2012). Studio-based approach let students construct, iteratively refine, and critically review the artifacts designed and developed by students which also increases the motivation and engagement level of students (Narayanan et al., 2012). However, studio-based collaborations are not enough for out of the classroom collaboration. Support-buddies (see figure 5.10 below for visualization) could be an alternative strategy to group project, peer programming, and studio-based approach, especially in introductory programming courses. In that strategy, students should be paired with two or more of their friends as in group projects. As an alternative to group project, each student would develop their own project. However, they would still be responsible for helping out their "buddies" and check the progress of each other's projects. In this way, every student could learn about teamwork, as well as embracing the sole responsibility of a project. "Especially in the context of practical tasks, paired or collaborative work and 'peer learning' has also been shown to be beneficial" (Robins et al., 2003, p. 158). Falkner and Falkner (2012) reported that required skills for programming such as problem-solving, communication, and critical thinking effectively and efficiently are developed through social constructivist activities. Through the practical tasks and development of the project, support-buddies could help students to overcome their problems with peer help. Peer support could also be helpful to the students who are shy about asking "simple" questions to the instructor as the results of this study showed. However, a similar problem about pair programming could also be the case for support-buddies. As Sanders (2002) reported that the stronger student in the pair frustrated to explain to the weaker student. Having more students than pair programming in each group and communication among groups could help to solve this problem. Wiedenbeck (2005) also reported that pairing students with different levels of self-efficacy is beneficial for first-time learners, and observing the performance of peers is a self-efficacy strengthening activity. Strategies similar to pair programming is most effective when the members of each pair have roughly same skill level and they meet in scheduled laboratory sessions.

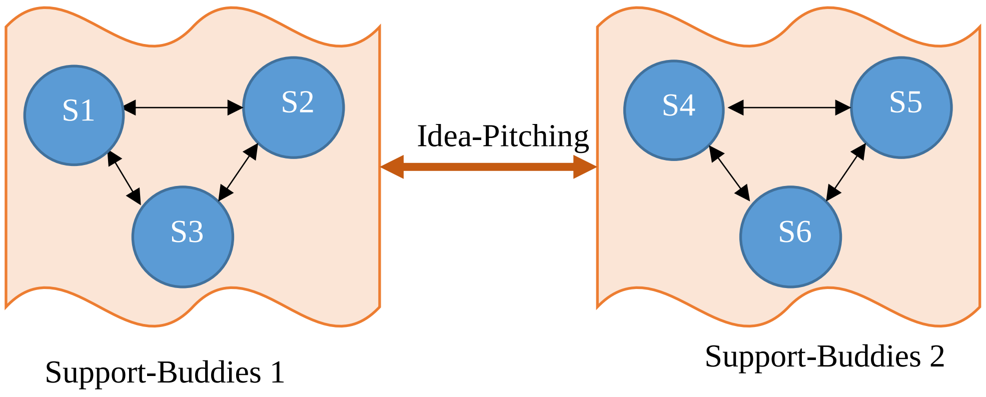


Figure 5.7 Peer Help/Idea-Pitching with Support-Buddies Strategy

Support-buddies strategy does incorporate five principles of cooperative learning that Mentz et al. reported: (1) Positive interdependence, (2) Individual accountability, (3) Face-to-face interaction, (4) Development of good social skills, (5) Group processing (Mentz et al., 2008, pp. 249–250).

Cooperative learning does also need a communication environment that allows students to interact an communicate with each other, which emphasize the importance of using an effective communication environment as it was mentioned before. Students had some problems when they needed to came up with a project idea. Additionally, when students faced with a problem or need a mentor to improve their projects, they need a discussion environment rather than a mentor. Even though the Facebook group was active in terms of idea sharing, an idea-sharing place in real life was also demanded by the students. Johnson and Caristi (2002) reported that communication among students helped them getting multiple ideas, and to avoid unnecessary trial and error. Novice programmers spend very little time planning the program they will create (Robins et al., 2003). Idea-pitching strategy could also improve their planning time since they are going to discuss what they will develop, and how they will be doing it. It could be helpful for students to use a weekly idea-pitching and recitation hour similar to the studio approach. Every week students and the instructor could meet and walk through the application they made during lab hours. Additionally, they could come up with the ideas for their final project and develop it theoretically with the help of their friends in a studio-like environment. Functions of the block used at the current week's lab could be explained step by step to re-emphasize the role of each block in recitation hour.

5.2.6 Misconceptions and Difficult Concepts

"Acquiring and developing knowledge about programming is a highly complex process" (Rogalski & Samurçay, 1990, p. 170). Students have had difficulty while learning some of the fundamental concepts of programming. Investigating the difficulties of the concepts could shed a light on the reasons and the possible solutions. Some strategies were developed to overcome the difficult concepts, however, it should be noted that there would be no single solution for every learner. The background of the students heavily affects the possible strategy to be implemented. One of the findings about the concepts was the same name with different features could lead to confusion in students. Variables and Clock concepts were among them. As it was mentioned in-detail, variable concept is taught students through the mathematics course before programming. Difference in the conceptual meaning and usage could end with a misconception of students. Clock concept in programming is a loop with changeable interval, while clock normally has a fixed interval by 1 second. The concepts and possible strategies to overcome the difficulty explained below.

One of the essential programming concept that students found difficult to understand was variables. Rogalski and Samurçay (1990) stated that variables are more complex than it seems, which could lead to diverse problems. Freiermuth, Hromkovič, and Steffen (2008) reported that starting with the variable concept too early could lead to a confusion among students. Especially students from the elementary mathematics education program had a misconception about it since the exact name was used for mathematical variables. In mathematical variables, they were asked to find the value of the variable. On the other hand, programming variable was determined by them and changed through the calculations. Defining the variable and assigning the first value were an unfamiliar concept for them. While the name stays the same throughout the application, value changes constantly (Rogalski & Samurçay, 1990). Robins et al.

(2003) also reported that updating and testing the variables are equally complex for novice programmers, but the initialization of the variable is more difficult for the learners. Instructor should clearly specify and emphasize the difference between a mathematical variable and programming variable with simple examples at the beginning of the concept. The examples should solely on variable use rather than being integrated into an example with more than one aspect. After the simple examples, usage and role of the variables should be emphasized constantly at the following weeks. Sajaniemi et al. (2006) offered to assign roles for variables, it could be confusing since the roles are too complicated for non-majors. Instead of that strategy appropriate naming of the variables could be used as a strategy for novice programmers. Students stated that after implementing the variables without tutorials, they understood it better. Simple examples with appropriate naming might be a simple but useful strategy for the first-time learners. Complex topics like databases, arrays (lists), variables could be used in multiple simple examples to help students to understand the concept thoroughly. Complex programs could broken down into mini programs as manageable tasks to help students understand the parts of a complex problems (Lye & Koh, 2014). Examples and tutorials are a good source of information to look back and remember the concepts. Therefore, the concepts they had could be listed in an index to help students to find what they need.

Another strategy to overcome the difficulties of programming concepts is using analogies from the real world. It is very easy to use the real clock to teach the clock concept. However, changing the clock interval could confuse the students, when the interval is more or less from one second. With basic examples like countdown timer, and heartbeat application concept of clock and interval could be learned more easily. Programming concepts without real-life analogies are difficult to understand for novice programmers (Buitrago Flórez et al., 2017). During the course hours, students stated that analogies are helpful for them to understand clock concept better. Especially after seeing it in a clock-focused example. Due to the general game concept of the examples clock concept was used multiple times. As expected students embraced it easier than the other difficult concepts. Complex topics could also be integrated into the projects of the students as smaller modules since most of the students stated that they learned the harder concepts while they were developing their projects.

Homework-like tasks related to the projects of the students could help them to understand the programming of the concepts of the week. If the concept is not related to their project, it could be used as a practice of the concept. Possible points and strategies to be considered to overcome the difficulties that were stated by students can be seen below in figure 5.11.

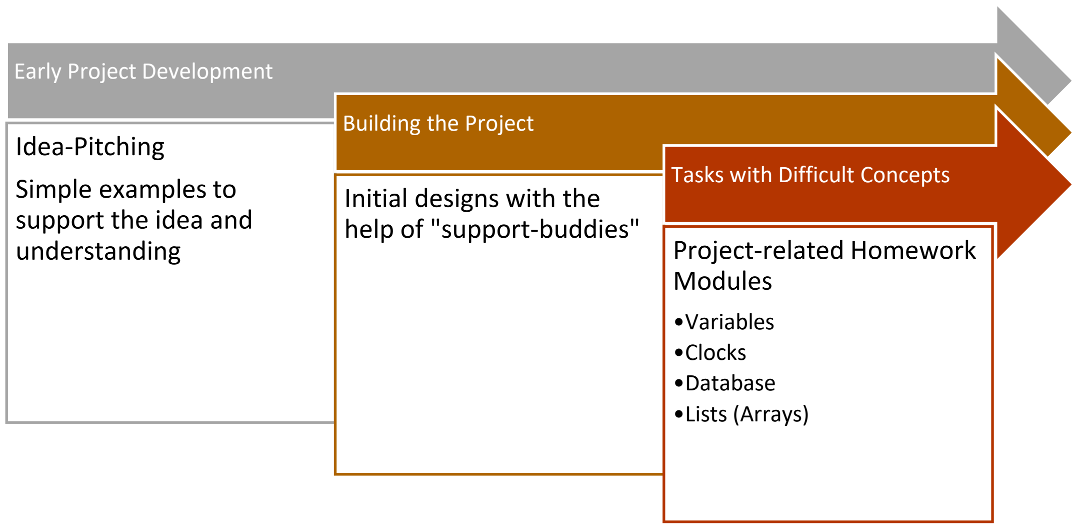

Early Project Development
Idea-Pitching
Simple examples to support the idea and understanding
Building the Project
Initial designs with the help of "support-buddies"
Tasks with Difficult Concepts
Project-related Homework Modules
•Variables
•Clocks
•Database
•Lists (Arrays)

Figure 5.8 Strategies to Overcome Difficulties of Students

5.2.7 Learner-Centered Formative Assessment

Traditionally assessment’s primary role is evaluating the comprehension of factual knowledge of the students (Webber & Tschepikow, 2013). Assessment can be used for (a) giving grades, (b) give feedback to learners to support the learning process, (c) predicting the future learning and success of the learners (Dagiene & Skupas, 2013, p. 82). Traditional assessment method in programming by scoring the source codes and emphasizing the wrong points may not be helpful to students since the reason behind errors could be the concepts that student did not fully comprehend (Phuong, 2010). On the other hand, learner-centered assesment is seen as an activity to foster student learning by (1) providing constructive and progressive feedback to students, (2) having students to evaluate each other’s work, (3) servicing assignments that require community interaction. As the design of the course should embrace an eclectic and formative approach, assessment should embrace a learner-centered formative assessment type. The main evaluation instrument should be the products of the students. Even the theoretical information should be evaluated from actual programs relevant to the product-first approach. Learner-centered assessment includes a mechanism for prompt feedback, foster collaboration and increased communication of students and faculty, and should require students to demonstrate meaningful application of knowledge and skills to real-world tasks; this would lead to concrete evidence of student's development of knowledge in which students should create meaningful applications for real-world tasks (Jon Mueller, 2005; Webber & Tschepikow, 2013). Learning-centered assessment which is more relevant for learning-centered education could increase the student-student and student-faculty interaction (Webber & Tschepikow, 2013) as this study suggested in and out of the classroom, face-to-face and through a communication medium. Of course, use of learner-centered assessment does not ensure higher performance (Webber & Tschepikow, 2013), it should be applied with other instructional strategies that support learning as this study suggests. Weber and Tschepikow (2013) reported that according to many research studies, learner-centered assessment is the best practice to evaluate students in higher education. It was more beneficial for learners to be given the opportunity to demonstrate their knowledge and skills in multiple ways such as presenting the projects, quizzes, open-ended questions (Grover, 2017).

Grover (2017) stated that presenting their projects worked better than summative tests for all of the students including the ones performed poorly in the summative test. To improve the effectiveness, efficiency, and engagement level of the course, instructor should constantly evaluate the performance of the students through the products, rather than summatively evaluate them to give grades. Additionally, as students stated, homework could also be a source of effectiveness to learn the concepts, getting feedback from the instructor and their peers. Open and formative feedback by using a communication medium was also one of the factors that students efficiently benefited from. As the summative evaluation, students found that presenting their projects by explaining the process are more beneficial for their understanding.

5.3 Conclusion and Prescription for the Future Courses

This study suggested strategies and recommendations that could help instructors to design an effective, efficient, and engagement introductory programming course. The strategies offered extracted from process of designing and developing an introductory programming course based on the experiences of both the instructor and the learners.

It is a fact that each course is unique regarding its learners, instructor, environment etc. The strategies and recommendations offered in this study are not universal. However, they could be still helpful to build an introductory programming course, since the strategies and recommendations followed were extracted from the experiences, opinions, and observations obtained through the process of course development and design cycles. The instructional strategies and recommendations presented below given with their expected effect on the course.

1- **Knowing the learners and defining their characteristics before designing the course:** Even though there is no need to conduct a research study, data should be collected through a survey or any other form that could reveal (a) programming experience, (b) the perception and attitude towards programming (c) demographic information, (d) the purpose of taking the course, (e) the communication medium choice of the students.

2- Searching and choosing the programming environment(s) (engagement, efficiency, and effectiveness): Programming environments are crucial for the course since the capabilities of the environments shapes the curriculum of the course. The study put forward that programming environment to use in an introductory programming course should have the following features

   a. Easy to use, to develop, and to learn
   b. Visual to see the product, to see the process, and to see the mistakes
   c. Helpful to prevent mistakes, to immediately show the correct way, by providing scaffolding
   d. Capable to be connected students' world, to develop appealing products

   Depending on the characteristics of the learners an easy to learn textual environment could be chosen as the supporting environment to let learners see the underlying structure and be familiar with textual programming to ease the transition.

3- Searching and choosing the communication medium (engagement, efficiency, and effectiveness): The communication medium is one of the essential parts of the study. Choosing a communication medium which is

   a. Popular and commonly used among students
   b. Notifying the learners instantly

    c. Multi-directional, interactive, and allowing open communication
    d. Allowing to be used as a resource hub

4- **Designing the course activities according to product-first approach in which examples and assignments should be (effectiveness, and engagement):** The course activities should be designed as the products at the center. The products of the tutorials, examples, and assignments should be
    a. Visual and Concrete
    b. Purposive and functional
    c. Meaningful/related to life
    d. Products with basic algorithms integrated
    e. Relevant to Students

5- **Using spiral approach with a course theme for the course flow (effectiveness, and engagement):** The course should be designed according to spiral approach in which the basic concepts introduced and expanded through increasing difficulty. Presenting the same concepts with increasing difficulty and adding new concepts on top of the previous knowledge. Additionally, spiral approach and product-first approach appeared to work better if the course was designed according to a theme. In this study games were chosen based on the interest of the students, and relevancy of gaming concepts to the course. A different concept could be chosen, if the interest of the audience is different.

6- **Presenting the examples with both top-down and bottom-up approach (effectiveness, and efficiency):** Main two approach in programming learning are top-down and bottom-up approach. According to instructional strategies such as spiral approach and above, bottom-up approach is more relevant to use. Additionally, bottom-up approach is also better for introductory level. However, results showed that using top-down approach is also beneficial to understand different perspectives, debugging the programs, and see working parts as a whole.

7- **Promoting computational thinking through real-life examples and on-paper flowchart activities (effectiveness, and engagement):** Computational thinking is not the main curriculum in an introductory programming course, however, it could be emphasized by providing flowchart examples to students. Making the thinking styles explicit by connecting the daily routines and programming problems could be one strategy for help students to realize computational thinking. Even though computational thinking is realized by students unconsciously throughout the course, integrating programming problems into learners' world could be an effective strategy for learners.

8- **Using dynamic tutorials with develop-it-more activities (effectiveness, efficiency, and engagement):** As spiral approach suggested activities should be designed in a simple to complex order. However, knowledge level difference between learners could make students get out of the flow zone during the in-class activity. A possible solution for this could be providing students dynamic tutorials, in which scaffolding will be presented to students when they are having a problem rather than presenting them fully well-structured. In addition to having dynamic tutorials, develop-it-more activities is another intervention created during the course. For students who are ahead of others develop-it-more activities could be presented at the end of each tutorial. Develop-it-more activities kept the students who are more experienced or successful in the flow zone during the course.

9- **Enhancing the student-student and student-instructor out-of-classroom communication, and interaction by using a communication medium (effectiveness, and engagement):** Choosing a communication medium is essential for the course design. However, using it to enhance communication is more important. Instructor should use the communiation medium to help students colaborate with and learn from each other. An open communication and helping environment should be set up and encouraged to let students help each other.

10- **Enhancing the collaboration skills by using support-buddies approach (effectiveness, efficiency, engagement):** Support-Buddies approach was introduced as a future intervention for the course. A mix of studio approach, group projects, and pair programming was presented as support-buddies to enhance the collaboration through the course and the project. According to the approach, each student will have a separate personal project. Each student will also have buddies to be responsible to help. Additionally, all of the students will meet weekly to review each other's projects like in studio approach.

11- **Using a learner-centered formative assessment (effectiveness, efficiency, and engagement):** Assessment of the course should be relevant to product-first approach and spiral approach. Rather than traditional assessment which based on grading, using a learner-centered assessment that focuses on giving feedback to learners to improve their understanding. Through the assignments and projects, student should evaluate and give feedback to each other, instead of having one-way feedback from instructor to students. It is also found that rather than having a final exam, it is more beneficial for students to present their projects and its process at the end of the course.

As it was mentioned before the strategies could be replaced or enhanced by the instructors. Flexible design of the course and implementation of the instructional strategies based on the context might be more helpful and successful since every course is unique regarding the components. Rogalski and Samurçay (1990) suggest using/developing "flexible strategies to derive benefits from programming aids (programming environment, programming methods)" (p.170). Soh, Samal, and Nugent (2007) each curriculum within a course related to CS should have effective, flexible, customizable, and modular components (p. 60). Robins et al. (2003) suggested opportunistic exploration for novice programmers where strategies are determined by episodes of problem-solving activities. Similarly being open for new strategies are essential for the instructional strategies. Even though, the strategies above based on the data from learners and instructor, the strategies presented above might not be relevant for all course contexts. Instructional strategies and recommendations could be benefitted, but the course should be designed flexible enough to be compatible with multiple environments, tools and strategies.

5.4 Future Studies

Those strategies are derived from an introductory course for non-CS major university students. Even though they depend on in-class experience. They needed to be tested and matured by other instructors and learners by implementing in-class. Other than strategies and recommendations suggested in this study, new strategies should be developed and added to the extracted ones. More in-class experience is needed and should be encouraged to make the experiences of different environments more explicit. In addition to the introductory course, a follow-up course could be designed to find if the knowledge acquired from this course could help students to advance in programming. Different programming environments which are suitable for advanced programming could be used with help of visualization tools and the strategies of this study. Integrating program visualization tools on a follow-up course to the introductory programming course could improve students' experiences and illustrate a positive outcome as some studies suggested (Bati et al., 2014).

The course constructed in this study have few students, therefore more follow-up studies are needed with crowded classes. Since one of the important challenges in education is presenting the information to a large audience, effectiveness of recommendations and strategies should be implemented to classes with different number of students.

There is a demand for computational thinking, and programming education to be integrated into K-12 curriculum (Brasiel et al., 2017; Julie Mueller, Beckett, Hennessey, & Shodiev, 2017; Pan, 2017). Similar studies could be conducted to see if the strategies are relevant for K-12 students and whether there is new instructional strategies and recommendations emerge from the studies.

Another aspect to focus on in future studies is using different data types from larger number of participants. Data of this study based on qualitative data from few participants. It was beneficial for a exploratory research design to inductively construct proncipiles. However, it would be also beneficial to test the effectiveness of the strategies and recommendations by collecting and analyzing quantitative data from different groups of students after implementing the strategies and recommendations.

REFERENCES

Abelson, H. (2009). App Inventor for Android. *Google Research Blog*. Retrieved from https://research.googleblog.com/2009/07/app-inventor-for-android.html

Al-Bow, M., Austin, D., Edgington, J., Fajardo, R., Fishburn, J., Lara, C., … Meyer, S. (2009). Using game creation for teaching computer programming to high school students and teachers. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education - ITiCSE ’09*, 104. https://doi.org/10.1145/1562877.1562913

AlHumoud, S., Al-Khalifa, H. S., Al-Razgan, M., & Alfaries, A. (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*, (April), 173–177. https://doi.org/10.1109/EDUCON.2014.6826086

Anderson, J. R. (2015). *Cognitive psychology and its implications* (Eigth Edit). New York, NY: Worth Publishers.

Anderson, J. R., & Skwarecki, E. (1986). The automated tutoring of introductory computer programming. *Communications of the ACM, 29*(9), 842–849. https://doi.org/10.1145/6592.6593

Anfara, V. A., & Mertz, N. T. (2006). *Theoretical Frameworks in Qualitative Research*. Thousand Oaks, CA: SAGE Publications, Inc. https://doi.org/10.1017/CBO9781107415324.004

Apiola, M., & Tedre, M. (2012). New perspectives on the pedagogy of programming in a developing country context. *Computer Science Education*, *22*(3), 285–313. https://doi.org/10.1080/08993408.2012.726871

Araujo, L. G. J., Bittencourt, R. A., & Santos, D. M. B. (2018). Contextualized Spiral Learning of Computer Programming in Brazilian Vocational Secondary Education. In *Proceedings of the 48th Annual Frontiers In Education Conference*. San Jose, California.

Armoni, M., & Gal-Ezer, J. (2014). High school computer science education paves the way for higher education: the Israeli case. *Computer Science Education*, *24*(2–3), 101–122. https://doi.org/10.1080/08993408.2014.936655

Barab, S., & Squire, K. (2004). Design-Based Research: Putting a Stake in the Ground. *Journal of the Learning Sciences*, *13*(1), 1–14. https://doi.org/10.1207/s15327809jls1301_1

Barg, M., Fekete, A., Greening, T., Hollands, O., Kay, J., Kingston, J. H., & Crawford, K. (2000). Problem-Based Learning for Foundation Computer Science Courses. *Computer Science Education*, *10*(2), 109–128. https://doi.org/10.1076/0899-3408(200008)10:2;1-C;FT109

Bati, T. B., Gelderblom, H., & van Biljon, J. (2014). A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa. *Computer Science Education*, *24*(1), 71–99. https://doi.org/10.1080/08993408.2014.897850

Ben-Ari, M. (2013). Visualization of programming. In D. M. Kadijevich, C. Angeli, & C. Schulte (Eds.), *Improving Computer Science Education* (pp. 52–65). New York, NY: Routledge. https://doi.org/10.4324/9780203078723-14

Bennedsen, J., & Caspersen, M. E. (2012). Persistence of elementary programming skills. *Computer Science Education*, *22*(2), 81–107. https://doi.org/10.1080/08993408.2012.692911

Berg, B. L. (2001). *Qualitative research methods for the social sciences*. *Qualitative Research* (Vol. Seventh Ed). https://doi.org/10.2307/1317652

Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. In P. Romero, J. Good, E. Acosta Chaparro, & S. Bryant (Eds.), *Psychology of Programming Interest Group 17th annual workshop (PPIG 2005)* (pp. 293–305). Brighton.

Bernard, M., & Bachu, E. (2015). Enhancing the Metacognitive Skill of Novice Programmers Through Collaborative Learning. In *Metacognition: Fundaments, Applications, and Trends. Intelligent Systems Reference Library, vol 76* (Vol. 76, pp. 277–298). Springer, Cham. https://doi.org/10.1007/978-3-319-11062-2

Bertea, A. F. (2011). Mobile Learning Applications Using Google App Inventor For Android. In *The 7th International Scientific Conference eLearning and Software for Education*. Bucharest.

Bogdan, R. C., & Biklen, S. K. (2007). *Qualitative Research for Education: An Introduction to Theory and Methods*. Boston, MA: Pearson Education Inc. https://doi.org/10.1177/1468794107085301

Bosch, N., & D'Mello, S. (2015). The Affective Experience of Novice Computer Programmers. *International Journal of Artificial Intelligence in Education*, *27*, 181–206. https://doi.org/10.1007/s40593-015-0069-5

Boyer, K. E., Phillips, R., Wallis, M. D., Vouk, M. a., & Lester, J. C. (2009). Investigating the role of student motivation in computer science education through one-on-one tutoring. *Computer Science Education*, *19*(2), 111–135. https://doi.org/10.1080/08993400902937584

Brasiel, S., Close, K., Jeong, S., Lawanto, K., Janisiewicz, P., & Martin, T. (2017). Measuring Computational Thinking Development with the FUN! Tool. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 327–347). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_20

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *2012 annual meeting of the AERA* (pp. 1–25). Vancouver, Canada.

Bruce, K. B. (2005). Controversy on how to teach CS 1. *ACM SIGCSE Bulletin, 37*(2), 111–117. https://doi.org/10.1145/1083431.1083477

Bryson, C., & Hand, L. (2007). The role of engagement in inspiring teaching and learning. *Innovations in Edcuation and Teaching International, 44*(4), 349–362.

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, *87*(4), 834–860. https://doi.org/10.3102/0034654317710096

Buss, A., & Gamboa, R. (2017). Teacher Transformations in Developing Computational Thinking: Gaming and Robotics Use in After-School Settings. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 189–203). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_12

Byrne, P., & Lyons, G. (2001). The Effect of Student Attributes on Success in Programming. *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education,* 49–52. https://doi.org/10.1109/ICETC.2009.35

Cambranes, E. (2013). Supporting novice programmers with natural language in the early stage of programming. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing* (pp. 173–174). IEEE. https://doi.org/10.1109/VLHCC.2013.6645271

Castelluccio, B. M. (2012). Want an App , Write an App, *Strategic Finance* (September, 2012).

Cetin, I. (2013). Visualization: a tool for enhancing students' concept images of basic object-oriented concepts. *Computer Science Education*, *23*(1), 1–23. https://doi.org/10.1080/08993408.2012.760903

Chao, C. (2006). An Investigation of Learning Style Differences and Attitudes toward Digital Game-based Learning among Mobile Users. In *2006 Fourth IEEE International Workshop on Wireless, Mobile and Ubiquitous Technology in Education (WMTE'06)*. IEEE. https://doi.org/10.1109/WMTE.2006.261340

Charlton, T., Devlin, M., & Drummond, S. (2009). Using Facebook to improve communication in undergraduate software development teams. *Computer Science Education*, *19*(4), 273–292. https://doi.org/10.1063/1.2756072

Cheong, Y. F., Pajares, F., & Oberman, P. S. (2004). Motivation and Academic Help-Seeking in High School Computer Science. *Computer Science Education*, *14*(1), 3–19. https://doi.org/10.1076/csed.14.1.3.23501

Christiansen, H. (2004). Teaching Computer Languages and Elementary Theory for Mixed Audiences at University Level. *Computer Science Education*, *14*(3), 205–234. https://doi.org/10.1080/0899340042000302727

Corbin, J. M., & Strauss, A. L. (2008). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Thousand Oaks, CA: Sage Publications.

Creswell, J. W. (2007). *Qualitative inquiry and research design: Choosing among five approaches*. Thousand Oaks, CA: SAGE Publications, Inc. https://doi.org/10.1111/1467-9299.00177

Creswell, J. W. (2009). *Research Design Qualitative, Quantitative, and Mixed Methods Approaches* (Vol. 3rd). Thousand Oaks, CA: SAGE Publications, Inc. https://doi.org/10.1016/j.math.2010.09.003

Creswell, J. W. (2012). *Educational Research: Planning, Conducting, and Evaluating Quantitaive and Qualitative Research*. Boston, MA: Pearson.

Creswell, J. W. (2014). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Thousand Oaks, CA: SAGE Publications, Inc.

Croff, C. H. (2017). Teaching Computational Thinking Patterns in Rural Communities. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 175–188). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_11

Csikszentmihalyi, M. (1991). *Flow: the psychology of optimal experience*. New York: HarperPerennial.

Dagiene, V., & Skupas, B. (2013). Assessment of students' programs. In D. M. Kadijevich, C. Angeli, & C. Schulte (Eds.), *Improving Computer Science Education* (pp. 82–97). New York, NY: Routledge. https://doi.org/10.4324/9780203078723-16

Dai, D. Y. (2012). *Design Research on Learning and Thinking in Educational Settings Enhancing Intellectual Growth and Functioning*. New York: Routledge.

Davies, S. P. (1993). Models and theories of programming strategy. *International Journal of Man-Machine Studies*, *39*(2), 237–267. https://doi.org/10.1006/IMMS.1993.1061

de Raadt, M. (2007). A Review of Australasian Investigations into Problem Solving and the Novice Programmer. *Computer Science Education*, *17*(3), 201–213. https://doi.org/10.1080/08993400701538104

Dede, C. (2005). Why design-based research is both important and difficult. *Educational Technology*, *45*(1 (Jan-Feb)), 5–8.

Deek, F. P., & McHugh, J. A. (1998). A survey and critical analysis of tools for learning programming. *Computer Science Education*, *8*(2), 130–178.

Demarco, T., & Lister, T. (2013). *Peopleware Productive Projects and Teams* (Third Edit). Indianapolis, IN: Addison-Wesley.

DeMarries, K. (2004). Qualitative Interview Stories: Learning Through Experience. In K. DeMarries & S. D. Lapan (Eds.), *Foundations for Research: Methods of Inquiry in Education and the Social Sciences*. Mahwah, N.J.: Lawrence Erlbaum Assoc.

Dennen, V. P., & Burner, K. J. (2017). Identity, context collapse, and Facebook use in higher education: putting presence and privacy at odds. *Distance Education*, *38*(2), 173–192. https://doi.org/10.1080/01587919.2017.1322453

Denzin, N. K., & Lincoln, Y. S. (1998). *The Landscape of Qualitative Research : Theories and Issues*. Thousand Oaks, CA: SAGE Publications, Inc.

du Boulay, B. (1989). Some difficulties of learning to program. In E. M. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer* (pp. 283–299). Hillsdale, NJ: Lawrence Erlbaum Assoc.

Effective. (n.d.). Retrieved September 8, 2018, from https://www.merriam-webster.com/dictionary/effective

Efficient. (n.d.). In *Merriam-Webster Dictionary*. Retrieved from https://www.merriam-webster.com/dictionary/efficient

Ellison, N. B., Steinfield, C., & Lampe, C. (2007). The Benefits of Facebook "Friends:" Social Capital and College Students' Use of Online Social Network Sites. *Journal of Computer-Mediated Communication*, *12*(4), 1143–1168. https://doi.org/10.1111/j.1083-6101.2007.00367.x

Engaging. (n.d.). In *Merriam-Webster Dictionary*. Retrieved from https://www.merriam-webster.com/dictionary/engaging

Falkner, K., & Falkner, N. J. G. (2012). Supporting and Structuring "Contributing Student Pedagogy" in Computer Science Curricula. *Computer Science Education*, *22*(4), 413–443. https://doi.org/10.1080/08993408.2012.727713

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2004). The TeachScheme! Project: Computing and Programming for Every Student. *Computer Science Education*, *14*(1), 55–77. https://doi.org/10.1076/csed.14.1.55.23499

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. H. (2012). *How to Design and Evaluate Research in Education* (Vol. 53). New York, NY: McGraw-Hill.

Freiermuth, K., Hromkovič, J., & Steffen, B. (2008). Creating and Testing Textbooks for Secondary Schools. In *Informatics Education - Supporting Computational Thinking* (pp. 216–228). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69924-8_20

Gardner, H., & Davis, K. (2013). *The app generation: How today's youth navigate identity, intimacy, and imagination in a digital world*. New Haven, CT: Yale University Press.

Gardner, J. (2010). Applying Merrill's First Principles of Instruction: Practical methods based on a review of the literature. *Educational Technology*, *50*(2), 20–25. Retrieved from http://ezproxy.lib.indiana.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=eric&AN=EJ890773&site=ehost-live%5Cnhttp://asianvu.com/bookstoread/etp/

Gasparinatou, A., & Grigoriadou, M. (2011). Supporting students' learning in the domain of computer science. *Computer Science Education*, *21*(1), 1–28. https://doi.org/10.1080/08993408.2010.509909

Gibbs, G. R. (2007). *Qualitative Research kit: Analyzing qualitative data*. London: SAGE Publications Ltd. https://doi.org/10.4135/9781849208574

Good, J., & Howland, K. (2017). Programming language, natural language? Supporting the diverse computational activities of novice programmers. *Journal of Visual Languages & Computing*, *39*, 78–92. https://doi.org/10.1016/j.jvlc.2016.10.008

Govender, I. (2009). Computers & Education The learning context : Influence on learning to program. *Computers & Education*, *53*(4), 1218–1230. https://doi.org/10.1016/j.compedu.2009.06.005

Gross, P., & Kelleher, C. (2010). Non-programmers identifying functionality in unfamiliar code: Strategies and barriers. *Journal of Visual Languages and Computing*, *21*(5), 263–276. https://doi.org/10.1016/j.jvlc.2010.08.002

Grover, S. (2017). Assessing Algorithmic and Computational Thinking in K-12: Lessons from a Middle School Classroom. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 269–288). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_17

Grover, S., & Pea, R. (2013). Computational Thinking in K – 12 : A Review of the State of the Field, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Guzdial, M., & Forte, A. (2005). Design process for a non-majors computing course. *ACM SIGCSE Bulletin*, *37*, 361. https://doi.org/10.1145/1047124.1047468

Guzdial, M., & Soloway, E. (2002). Teaching the Nintendo generation to program. *Communications of the ACM*, *45*(4), 17–21. https://doi.org/10.1145/505248.505261

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, *21*(2), 135–173. https://doi.org/10.1080/08993408.2011.579808

Hawi, N. (2010). The exploration of student-centred approaches for the improvement of learning programming in higher education, *7*(9), 47–57.

Herman, G. L. (2012). Designing contributing student pedagogies to promote students' intrinsic motivation to learn. *Computer Science Education*, *22*(4), 369–388. https://doi.org/10.1080/08993408.2012.727711

Herrington, J., Mckenney, S., Reeves, T., & Oliver, R. (2007). Design-based research and doctoral students: Guidelines for preparing a dissertation proposal. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2007*, *2007*(2007), 4089–4097.

Homer, M., & Noble, J. (2017). Lessons in Combining Block-based and Textual Programming. *Journal of Visual Languages and Sentinent Systems*, *3*, 22–39.

Hooshyar, D., Ahmad, R. B., & Nasir, M. H. N. M. (2014). A framework for automatic text-to-flowchart conversion: A novel teaching aid for novice programmers. *Proceeding - 2014 International Conference on Computer, Control, Informatics and Its Applications: "New Challenges and Opportunities in Big Data", IC3INA 2014*, 7–12. https://doi.org/10.1109/IC3INA.2014.7042592

Howles, T. (2009). A study of attrition and the use of student learning communities in the computer science introductory programming sequence. *Computer Science Education*, *19*(910534876), 1–13. https://doi.org/10.1080/08993400902809312

Hsu, Y.-C., & Ching, Y. (2013). Mobile App Design for Teaching and Learning : Educators' Experiences in an Online Graduate Course. *The International Review of Research in Open and Distance Learning*, *14*(4), 117–139.

Hsu, Y.-C., Rice, K., & Dawley, L. (2012). Empowering educators with Google's Android App Inventor: An online workshop in mobile app design. *British Journal of Educational Technology*, *43*(1), E1–E5. https://doi.org/10.1111/j.1467-8535.2011.01241.x

Ichinco, M., & Louis, S. (2016). Suggesting and Supporting Examples for Novice Programmers. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 260–261).

Ichinco, M., Zemach, A., & Kelleher, C. (2013). Towards generalizing expert programmers' suggestions for novice programmers. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, (1054587), 143–150. https://doi.org/10.1109/VLHCC.2013.6645259

IDC: Smartphone OS Market Share. (2017). Retrieved from http://www.idc.com/promo/smartphone-market-share/os

Iqbal, S., Chowdhury, M. U., & Harsh, O. K. (2013). Mobile devices supported learning for novice programmers. In *2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)* (pp. 277–282). Lodz. https://doi.org/10.1109/ICeLeTE.2013.6644388

Jaime, A., Blanco, J. M., Domínguez, C., Sánchez, A., Heras, J., & Usandizaga, I. (2016). Spiral and Project-Based Learning with Peer Assessment in a Computer Science Project Management Course. *Journal of Science Education and Technology*, *25*(3), 439–449. https://doi.org/10.1007/s10956-016-9604-x

Jing, L., Cheng, Z., Wang, J., & Zhou, Y. (2011). A spiral step-by-step educational method for cultivating competent embedded system engineers to meet industry demands. *IEEE Transactions on Education*, *54*(3), 356–365. https://doi.org/10.1109/TE.2010.2058576

Johnson, D. H., & Caristi, J. (2002). Using Extreme Programming in the Software Design Course. *Computer Science Education*, *12*(3), 223–234. https://doi.org/10.1076/csed.12.3.223.8616

Jones, S., & Burnett, G. (2008). Spatial ability and learning to program. *Human Technology*, *4*(May), 47–61. https://doi.org/10.17011/ht/urn.200804151352

Kafai, Y. B., & Burke, Q. (2013). Computer Programming Goes Back to School. *Phi Delta Kappan*, *95*(1), 61–65. https://doi.org/10.1177/003172171309500111

Kafai, Y. B., & Burke, Q. (2017). Computational Participation: Teaching Kids to Create and Connect Through Code. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 393–405). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_24

Kalas, I., & Winczer, M. (2008). Informatics as a Contribution to the Modern Constructivist Education. In *Informatics Education - Supporting Computational Thinking* (pp. 229–240). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69924-8_21

Katz, S., Allbritton, D., Aronis, J., Wilson, C., & Soffa, M. Lou. (2006). Gender, achievement, and persistence in an undergraduate computer science program. *ACM SIGMIS Database*, *37*(4), 42. https://doi.org/10.1145/1185335.1185344

Kaurel, H. G. (2016). *Easing the Transition from Visual to Textual Programming*. Norwegian University of Science and Technology.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2011). Understanding Computational Thinking before Programming. *International Journal of Game-Based Learning*, *1*(3), 30–52. https://doi.org/10.4018/ijgbl.2011070103

Keller, J. M. (1987). Development and use of the ARCS model of motivational design. *Journal of Instructional Development*, *10*(1932), 2–10. https://doi.org/10.1002/pfi.4160260802

Kelly, A. E. (2004). Design Research in Education: Yes, but is it Methodological? *Journal of the Learning Sciences*, *13*(1), 115–128. https://doi.org/10.1207/s15327809jls1301

Kemp, S. (2017). *Digital in 2017: Global Review*. Retrieved from https://wearesocial.com/special-reports/digital-in-2017-global-overview

Kim, B., Kim, T., & Kim, J. (2013). Paper-and-Pencil Programming Strategy toward Computational Thinking for Non-Majors: Design Your Solution. *Journal of Educational Computing Research*, *49*(4), 437–459. https://doi.org/10.2190/EC.49.4.b

Kim, D.-K., Jeong, D., Lu, L., Debnath, D., & Ming, H. (2015). Opinions on computing education in Korean K-12 system: higher education perspective. *Computer Science Education*, *25*(4), 371–389. https://doi.org/10.1080/08993408.2016.1140409

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ System and its Pedagogy. *Computer Science Education*, *13*(4), 249–268. https://doi.org/10.1076/csed.13.4.249.17496

Kurkovsky, S. (2013). Mobile Game Development: Improving student engagement and motivation in introductory computing courses. *Computer Science Education*, *23*(2), 138–157. https://doi.org/10.1080/08993408.2013.777236

Kwon, D., Yoon, I., & Lee, W. (2011). Design of Programming Learning Process using Hybrid Programming Environment for Computing Education, *5*(10), 1799–1813. https://doi.org/10.3837/tiis.2011.10.007

Langrich, M., & Schulze, J. (2015). Rethinking task types for novice programmers. In *Proceedings - Frontiers in Education Conference, FIE*. https://doi.org/10.1109/FIE.2014.7044421

Lee, Y. (2011). Scratch: Multimedia Programming Environment for Young Gifted Learners. *Gifted Child Today*, *34*(2), 26–31.

Levy, R. B., & Ben-ari, M. (2009). Adapting and merging methodologies in doctoral research. *Computer Science Education*, *19*(2), 51–67. https://doi.org/10.1080/08993400902937550

Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education*, *21*(2), 105–134. https://doi.org/10.1080/08993408.2011.579805

Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic Inquiry*. Newbury Park, CA: SAGE Publications, Inc.

Lister, R., Seppälä, O., Simon, B., Thomas, L., Adams, E. S., Fitzgerald, S., … Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In *Working group reports from ITiCSE on Innovation and technology in computer science education - ITiCSE-WGR '04* (Vol. 36, pp. 119–150). New York, NY: ACM Press. https://doi.org/10.1145/1044550.1041673

Lo, C. K., & Hew, K. F. (2017). Using "First Principles of Instruction" to Design Mathematics Flipped Classroom for Underperforming Students. *Educational Technology & Society, 20*(1), 222–236. https://doi.org/10.18178/ijlt.3.2.82-89

Long, J. (2007). Just For Fun: Using Programming Games in Software Programming Training and Education-A Field Study of IBM Robocode Community. *Journal of Information Technology Education, 6*, 279–290. https://doi.org/Article

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61. https://doi.org/10.1016/j.chb.2014.09.012

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57–80. https://doi.org/10.1080/08993408.2011.554722

MacGregor, S. K. (1988). Computer Programming Instruction. *Journal of Research on Computing in Education*, *21*(2), 155–164. https://doi.org/10.1080/08886504.1988.10781868

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, *39*(1), 223. https://doi.org/10.1145/1227504.1227388

Maleko, M., Hamilton, M., & D'Souza, D. (2012). Access to mobile learning for novice programmers via social networking sites. *Computer Science & Education (ICCSE), 2012 7th International Conference On*, (Iccse), 1533–1538. https://doi.org/10.1109/ICCSE.2012.6295355

Maleko, M., Hamilton, M., D'Souza, D., & Scholer, F. (2014). Understanding and analysing novice programmer interactions in a facebook programming group. In *Proceedings - 2014 International Conference on Teaching and Learning in Computing and Engineering, LATICE 2014* (pp. 112–119). https://doi.org/10.1109/LaTiCE.2014.28

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2013). A Holistic Framework for the Development of an Educational Game Aiming to Teach Computer Programming. *Proceedings of the 7Th European Conference on Games Based Learning, Vols 1 and 2*, 359–368.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment, *10*(4), 1–15. https://doi.org/10.1145/1868358.1868363.http

Manasijević, D., Živković, D., Arsić, S., & Milošević, I. (2016). Exploring students' purposes of usage and educational usage of Facebook. *Computers in Human Behavior, 60*, 441–450. https://doi.org/10.1016/j.chb.2016.02.087

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education*, *16*(3), 211–227. https://doi.org/10.1080/08993400600912384

Margulieux, L. E., Catrambone, R., & Guzdial, M. (2016). Employing subgoals in computer programming education. *Computer Science Education*, *26*(1), 44–67. https://doi.org/10.1080/08993408.2016.1144429

Marlowe, B. A., & Page, M. L. (2005). *Creating and sustaining the constructivist classroom*. Thousand Oaks, CA: Corwin Press.

Mason, J. (2002). *Qualitative researching*. London: SAGE Publications, Inc. https://doi.org/10.1016/S0143-6228(97)90005-9

Mason, R., & Cooper, G. (2013). Mindstorms robots and the application of cognitive load theory in introductory programming. *Computer Science Education*, *23*(4), 296–314. https://doi.org/10.1080/08993408.2013.847152

McKenney, S., & Reeves, T. (2012). *Conducting Educational Design Research*. New York: Routledge, Taylor & Francis Group.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(September), 239–264. https://doi.org/10.1080/08993408.2013.832022

Mendes, A. J., Paquete, L., Cardoso, A., & Gomes, A. (2012). Increasing student commitment in introductory programming learning. In *2012 Frontiers in Education Conference Proceedings* (pp. 1–6). Seattle, WA: IEEE. https://doi.org/10.1109/FIE.2012.6462486

Mentz, E., van der Walt, J. L., & Goosen, L. (2008). The effect of incorporating cooperative learning principles in pair programming for student teachers. *Computer Science Education*, *18*(4), 247–260. https://doi.org/10.1080/08993400802461396

Merriam, S. B. (1998). *Qualitative Research and Case Study Applications in Education*. San Francisco, CA: Jossey-Bass.

Merriam, S. B. (2009). *Qualitative research: A guide to design and implementation*. San Francisco, CA: Jossey-Bass. https://doi.org/10.1017/CBO9781107415324.004

Merrill, M. D. (2002). First principles of instruction. *Educational Technology, Research and Development, 50*(3), 43–59. https://doi.org/10.1007/BF02505024

Merrill, M. D. (2008). Converting e3-learning to e^3-learning: An alternative instructional design method. In S. Carliner & P. Shank (Eds.), *The E-Learning Handbook: Past Promises, Present Challenges* (pp. 359–400). San Francisco, CA: Pfeiffer.

Merrill, M. D. (2013). *First Principles of Instruction*. San Francisco, CA: Pfeiffer.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. Thousand Oaks, CA: SAGE Publications, Inc.

MIT App Inventor. (2017). MIT App Inventor. Retrieved September 7, 2017, from appinventor.mit.edu/explore/

Mozelius, P., Shabalina, O., Malliarakis, C., Tomos, F., Miller, C., & Turner, D. (2013). Let the Students Contruct Their own fun And Knowledge - Learning to Program by Building Computer Games. In *Proceedings of the 7th European Conference on Games Based Learning* (Vol. 1,2, pp. 418–426).

Mueller, J. (2005). The Authentic Assessment Toolbox: Enhancing Student Learning Through Online Faculty Development. *Journal of Online Learning and Teaching, 1*(1). Retrieved from http://jolt.merlot.org/vol1_no1_mueller.htm

Mueller, J., Beckett, D., Hennessey, E., & Shodiev, H. (2017). Assessing Computational Thinking Across the Curriculum. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 251–267). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_16

Narayanan, N. H., Hundhausen, C., Hendrix, D., & Crosby, M. (2012). Transforming the CS classroom with studio-based learning. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12* (pp. 165–166). New York, NY: ACM Press. https://doi.org/10.1145/2157136.2157188

Noone, M., & Mooney, A. (2017). First Programming Language : Visual or Textual? In *International Conference on Engaging Pedagogy (ICEP).*

Orfanakis, V., & Papadakis, S. J. (2014). Teaching basic programming concepts to novice programmers in Secondary Education using Twitter, Python, Arduino and a coffee machine. In *Proceedings of the 2014 Workshop on Interaction Design in Educational Environments*. https://doi.org/10.1007/978-3-319-55553-9

Pan, T.-Y. (2017). Reenergizing CS0 in China. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 351–362). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_21

Papert, S. (1993). *Mindstorms : children, computers, and powerful ideas*. New York: Basic Books.

Parhami, B. (2008). A puzzle-based seminar for computer engineering freshmen. *Computer Science Education*, *18*(4), 261–277. https://doi.org/10.1080/08993400802594089

Patton, M. Q. (2002). *Qualitative research and evaluation methods* (Vol. 3rd). Thousand Oaks, CA: SAGE Publications, Inc. https://doi.org/10.2307/330063

Phuong, D. D. (2010). Graining and Filling Understanding Gaps for Novice Programmers. In *2010 International Conference on Education and Management Technology (ICEMT 2010)* (pp. 60–64).

Pintrich, P. R. (2003). A Motivational Science Perspective on the Role of Student Motivation in Learning and Teaching Contexts. *Journal of Educational Psychology*, *95*(4), 667–686. https://doi.org/10.1037/0022-0663.95.4.667

Pokress, S. C., & Veiga, J. J. D. (2013). MIT App Inventor: Enabling Personal Mobile Computing, 3. Computers and Society; Human-Computer Interaction. https://doi.org/10.1145/2721914.2721935

Polit, D. F., & Beck, C. T. (2003). *Nursing Research Principles and Methods*. Phil: Lippincott Williams & Wilkins.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in Introductory Programming: What Works? *Communications of the ACM*, *56*(8), 34–36. https://doi.org/10.1145/2492007

Postholm, M. B., & Madsen, J. (2006). The Reseaercher’s Role: An Ethical Dimension. *Outlines*, *7*(1), 49–60.

Prensky, M. (2001). Digital Natives, Digital Immigrants. *On the Horizon*, *9*(5), 1–6. https://doi.org/10.1108/10748120110424816

Quaye, A. M., & Dasuki, S. I. (2017). A Computational Approach to Learning Programming Using Visual Programming in a Developing Country University. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 121–134). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_8

Reeves, T. C. (2006). Design Research from a Technology Perspective. In K. van den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.), *Educational design research* (pp. 52–66). London: Routledge.

Reigeluth, C. M. (1999a). The elaboration theory: Guidance for scope and sequence decisions. In C. M. Reigeluth (Ed.), *Instructional-Design Theories and Models: A New Paradigm of Instructional Theory* (pp. 425–454). New Jersey: Lawrence Erlbaum Assoc.

Reigeluth, C. M. (1999b). What is Instructional-Design Theory and How Is It Changing? In *Instructional Design Theories and Models: New Paradigms of Instructional Theory* (Vol. II, pp. 5–29).

Reiser, R. A. (2002). What field did you say you were in? Defining and naming our field. In R. A. Reiser & J. V. Dempsey (Eds.), *Trends and issues in instructional design and technology* (pp. 5–14). New Jersey: Merrill/Prentice Hall.

Repenning, A., Basawapatna, A. R., & Escherle, N. A. (2017). Principles of Computational Thinking Tools. In *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 291–305). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-52691-1_18

Resnick, M., Maloney, J., Monroy-, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch : Programming for All. *Communications of the ACM*, *52*(11), 60–67.

Richey, R. C. (1997). Research on instructional development. *Educational Technology Research and Development*, *45*(3), 91–100. https://doi.org/10.1007/BF02299732

Richey, R. C., & Klein, J. D. (2007). *Design and development research : methods, strategies, and issues*. Mahwah, N.J.: L. Erlbaum Associates.

Rist, R. S. (1995). Program structure and design. *Cognitive Science*, *19*(4), 507–561. https://doi.org/10.1016/0364-0213(95)90009-8

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Rogalski, J., & Samurçay, R. (1990). Acquisition of Programming Knowledge and Skills. In J. M. Hoc, T. R. G. Green, R. Samurçay, & D. J. Gillmore (Eds.), *Psychology of Programming* (pp. 157–174). London: Academic Press. https://doi.org/10.1016/B978-0-12-350772-3.50015-X

Rolandsson, L. (2013). Changing computer programming education: The dinosaur that survived in school: An explorative study about educational issues based on teachers' beliefs and curriculum development in secondary school. In *Proceedings - 2013 Learning and Teaching in Computing and Engineering, LaTiCE 2013* (pp. 220–223). https://doi.org/10.1109/LaTiCE.2013.47

Romeike, R. (2008). What's My Challenge? The Forgotten Part of Problem Solving in Computer Science Education. In *Informatics Education - Supporting Computational Thinking* (pp. 122–133). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69924-8_11

Rountree, J., Robins, A., & Rountree, N. (2013). Elaborating on threshold concepts. *Computer Science Education*, *23*(3), 265–289. https://doi.org/10.1080/08993408.2013.834748

Saito, D., & Yamaura, T. (2013). A new approach to Programming Language education for beginners with top-down learning. *International Journal of Engineering Pedagogy Is*, *3*(4), 16–21. https://doi.org/10.1109/TALE.2013.6654538

Sajaniemi, J. (2008). Special issue on psychology of programmiing. *Human Technology*, *4*(1).

Sajaniemi, J., Ben-Ari, M., Byckling, P., Gerdt, P., & Kulikova, Y. (2006). Roles of Variables in Three Programming Paradigms. *Computer Science Education, 16*(4), 261–279. https://doi.org/10.1080/08993400600874584

Sampaio, A., & Sampaio, I. (2012). Improving computing courses from the points of view of students and teachers: a review and an empirical study. *Computer Science Education*, *22*(2), 139–173. https://doi.org/10.1080/08993408.2012.692920

Sanders, D. (2002). Extreme Programming: The student view. *Computer Science Education*, *12*(3), 235–250. https://doi.org/10.1076/csed.12.3.235.8615

Sandoval-Reyes, S., Galicia-Galicia, P., & Gutierrez-Sanchez, I. (2011). Visual Learning Environments for Computer Programming. *2011 IEEE Electronics, Robotics and Automotive Mechanics Conference*, 439–444. https://doi.org/10.1109/CERMA.2011.76

Schäfer, A., Holz, J., Leonhardt, T., Schroeder, U., Brauner, P., & Ziefle, M. (2013). From boring to scoring – a collaborative serious game for learning and practicing mathematical logic for computer science education. *Computer Science Education*, *23*(2), 87–111. https://doi.org/10.1080/08993408.2013.778040

Schunk, D. H. (1981). Modeling and attributional effects on children’s achievement: A self-efficacy analysis. *Journal of Educational Psychology*, *73*(1), 93–105. https://doi.org/10.1037/0022-0663.73.1.93

Seneviratne, O. (2017). Making Computer Science Attractive to High School Girls with Computational Thinking Approaches: A Case Study. In P. J. Rich & C. B. Hodges (Eds.), *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 21–33). Cham: Springer. https://doi.org/10.1007/978-3-319-52691-1

Shneiderman, B. (1977). Teaching programming: A spiral approach to syntax and semantics. *Computers & Education*, *1*(4), 193–197. https://doi.org/10.1016/0360-1315(77)90008-2

Smutný, P. (2011). Visual programming for smartphones. In *Proceedings of the 2011 12th International Carpathian Control Conference, ICCC’2011* (pp. 358–361). https://doi.org/10.1109/CarpathianCC.2011.5945879

Soep, E. (2011). Youth Media. *National Civic Review*, *100*(3), 8–11. https://doi.org/10.1002/ncr.20073

Soh, L.-K., Samal, A., & Nugent, G. (2007). An integrated framework for improved Computer Science Education: Strategies, implementations, and results. *Computer Science Education*, *17*(1), 59–83. https://doi.org/10.1080/08993400701203782

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 850–858. https://doi.org/10.1145/6592.6594

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education*, *13*(2), 1–31. https://doi.org/10.1145/2483710.2483713

Sorva, J., Karavirta, V., & Malmi, L. (2013). A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education*, *13*(4), 1–64. https://doi.org/10.1145/2490822

Sorva, J., Lönnberg, J., & Malmi, L. (2013). Students' Ways of Experiencing Visual Program Simulation. *Computer Science Education*, *23*(3), 207–238. https://doi.org/10.1080/08993408.2013.807962

Spicer, J. C. (1983). A spiral approach to Software Engineering Project Management Education. *ACM SIGSOFT Software Engineering Notes*, *8*(3), 30–38. https://doi.org/10.1145/1010891.1010895

Statista. (2018a). Leading countries based on number of Facebook users as of July 2018 (in millions). Retrieved from https://www.statista.com/statistics/268136/top-15-countries-based-on-number-of-facebook-users/

Statista. (2018b). Number of daily active Facebook users worldwide as of 2nd quarter 2018 (in millions). Retrieved September 5, 2018, from https://www.statista.com/statistics/346167/facebook-global-dau/

Su, A. Y. S., Yang, S. J. H., Hwang, W.-Y., Huang, C. S. J., & Tern, M.-Y. (2014). Investigating the role of computer-supported annotation in problem-solving-based teaching: An empirical study of a Scratch programming pedagogy. *British Journal of Educational Technology*, *45*(4), 647–665. https://doi.org/10.1111/bjet.12058

Tangney, B., Oldham, E., Conneely, C., Barrett, S., & Lawlor, J. (2010). Pedagogy and Processes for a Computer Programming Outreach Workshop—The Bridge to College Model. *IEEE Transactions on Education*, *53*(1), 53–60. https://doi.org/10.1109/TE.2009.2023210

Tanner, A., & Duncan, S. (2013). On Integrating Mobile Applications into the Digital Forensic Investigative Process. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *4*(8), 56–61. Retrieved from http://ijacsa.thesai.org/

Thomas, P., King, D., & Minocha, S. (2009). The effective use of a simple wiki to support collaborative learning activities. *Computer Science Education*, *19*(4), 293–313. https://doi.org/10.1080/08993400903384943

Van den Akker, J. (1999). Principles and methods of development research. In J. Van den Akker, R. M. Branch, K. Gustafson, N. Nieven, & T. Plomp (Eds.), *Design aproaches and tools in education and training* (pp. 1–14). Dordrecht: Kluwer Academic Publishers.

Van den Akker, J., Gravemeijer, K., McKenney, S., & Nieveen, N. (2006). Introducing educational design research. In J. Van den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.), *Educational design research* (pp. 3–7). New York, NY: Routledge. https://doi.org/10.1111/j.1467-8535.2008.00855_1.x

Veladat, F., & Mohammadi, F. (2011). Spiral learning teaching method: Stair stepped to promote learning. *Procedia - Social and Behavioral Sciences*, *29*, 1115–1122. https://doi.org/10.1016/j.sbspro.2011.11.345

Walker, D. (2006). Toward productive design studies. In J. Van Den Akker, K. Gravemeijer, S. McKenney, & N. Nieveen (Eds.), *Educational design research* (pp. 8–13). New York, NY.

Webber, K. L., & Tschepikow, K. (2013). The role of learner-centred assessment in postsecondary organisational change. *Assessment in Education: Principles, Policy & Practice*, *20*(2), 187–204. https://doi.org/10.1080/0969594X.2012.717064

Wellman, B., Haase, A. Q., Witte, J., & Hampton, K. (2001). Does the Internet Increase, Decrease, or Supplement Social Capital? Social Networks, Participation, and Community Commitment. *American Behavioral Scientist*, *45*(3), 436–455. Retrieved from http://abs.sagepub.com

Wiedenbeck, S. (2005). Factors affecting the success of non-majors in learning to program. In *Proceedings of the 2005 international workshop on Computing education research - ICER '05* (pp. 13–24). New York, New York, USA: ACM Press. https://doi.org/10.1145/1089786.1089788

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197–212. https://doi.org/10.1076/csed.12.3.197.8618

Willman, S., Lindén, R., Kaila, E., Rajala, T., Laakso, M.-J., & Salakoski, T. (2015). On study habits on an introductory course on programming. *Computer Science Education*, *25*(3), 276–291. https://doi.org/10.1080/08993408.2015.1073829

Wilson, A., Hainey, T., & Connolly, T. (2012). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. *Proceedings of the European Conference on Games Based Learning*, (2007), 549–558.

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin*, *33*(1), 184–188. https://doi.org/http://doi.acm.org/10.1145/366413.364581

Wilson, M. E. (2004). Teaching, learning, and millennial students. *New Directions for Student Services*, *106*, 59–71. https://doi.org/10.1002/ss.125

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. https://doi.org/10.1145/1227504.1227378

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Winslow, L. E. (1996). Programming Pedagogy --A Psychological Overview. *ACM SIGCSE Bulletin*, *28*(3), 17–22. https://doi.org/10.1145/234867.234872

Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with Scratch in CS 1. In *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education* (pp. 2–3).

Wong, B. (2017). 'I'm good, but not that good': digitally-skilled young people's identity in computing. *Computer Science Education*, *26*(4), 299–317. https://doi.org/10.1080/08993408.2017.1292604

Wright, B. G. A., Rich, P., & Leatham, K. R. (2012). Curriculum, (April), 3–10.

Writer, J. (2017). Spiral Learning Applies To Writers. Retrieved September 5, 2018, from https://jubileewriter.wordpress.com/2017/08/17/spiral-learning-applies-to-writers/

Wyeld, T., & Barbuto, Z. (2014). Don't hide the code!: Empowering novice and beginner programmers using a HTML game editor. *Proceedings of the International Conference on Information Visualisation*, 125–131. https://doi.org/10.1109/IV.2014.56

Yadav, A., Gretter, S., Hambrusch, S., & Sands, P. (2017). Expanding computer science education in schools: Understanding teacher experiences and challenges. *Computer Science Education*, *26*(4), 235–254. https://doi.org/10.1080/08993408.2016.1257418

Yovcheva, B. B. (2008). Spiral Teaching of Programming to 10–11 Year-Old Pupils After Passed First Training (Based on the Language C++). In *Informatics Education - Supporting Computational Thinking* (pp. 171–179). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69924-8_16

APPENDIX A: INTERVIEW

GÖRÜŞME

Merhaba. Bu görüşmenin amacı bu dönem almış olduğunuz "Görsel programlama ile yazılım geliştirme" dersi ve öğrenme ortamı ile ilgili görüşlerinizi öğrenmektir. Bu ders ile sahip olduğunuz deneyimleri, yaşadığınız zorlukları ve önerilerinizi benimle paylaşırsanız çok sevinirim.

Sakıncası yoksa görüşmeyi kaydetmek istiyorum. Bu şekilde görüşmeden sonra yanıtlarınızı daha iyi analiz edebilirim. Bu görüşme boyunca söylediğiniz herşey gizli kalacaktır. Elde edilen bilgiler hiç kimseye iletilmeyecektir ve açıklanan hiçbir bilgi üzerinde isminiz belirtilmeyecektir. Çalışmaya katıldığınız için teşekkür ederim.

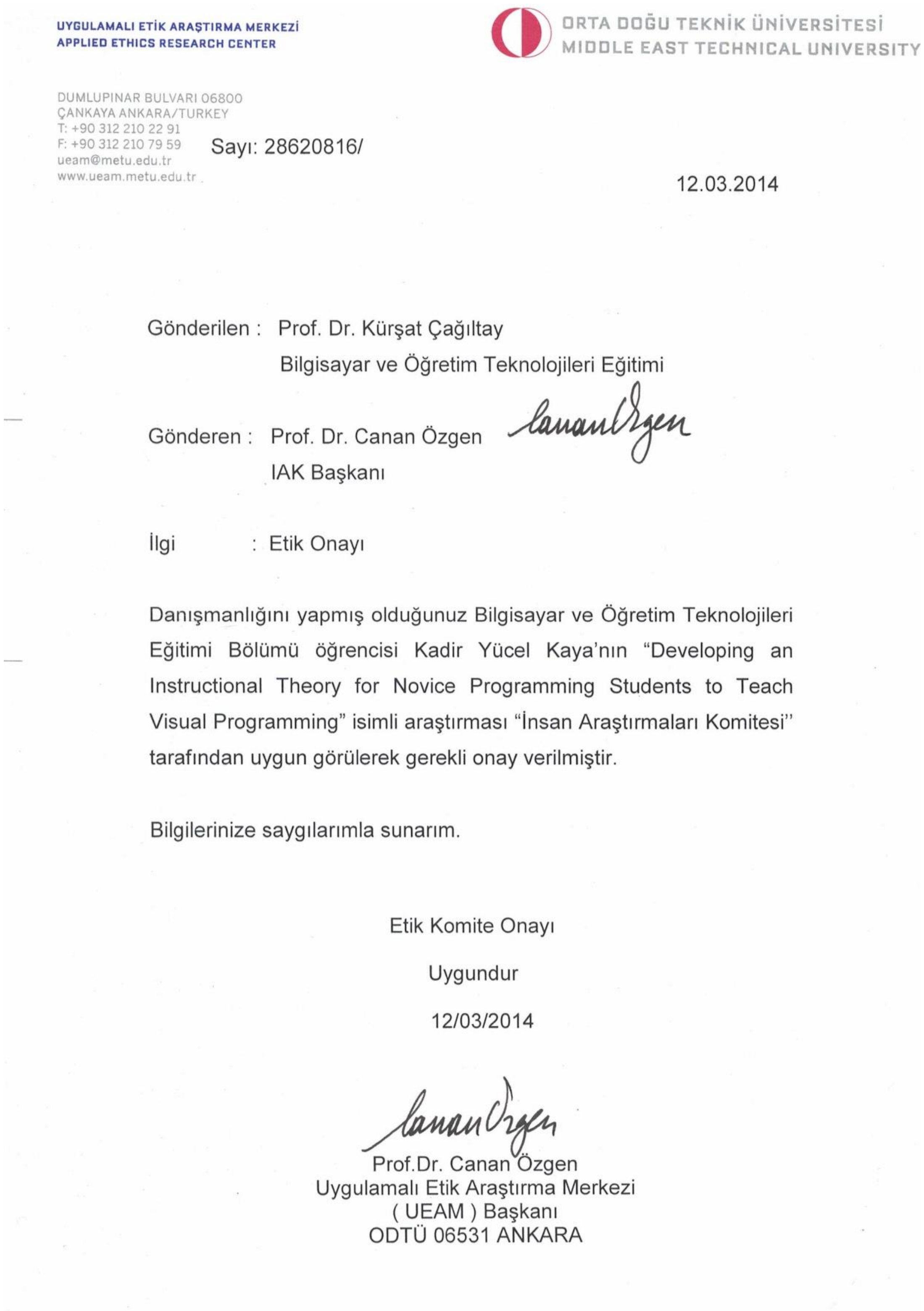
Çalışma sonucunda sizin deneyimlerinizden faydalanarak etkili bir programlama eğitimi tasarlamayı planlıyoruz. Öncelikle kendinizi tanıtır mısınız? Adınız, bölümünüz, sınıfınız…

1. Daha önce hiç programlama deneyiminiz oldu mu? Olduysa
    a) Hangi ortamı ve hangi programlama dilini kullandınız?
    b) Programlama öğrenirken zorluk yaşandınız mı?
        a. Evet ise yaşadığınız zorluklar nelerdir?
        b. Hayır ise dersin en çok hangi yönleri hoşunuza gitti?
    c) Bu dönem aldığınız programlama dersi ile karşılaştırdığınızda aklınıza gelen
        a. Olumlu yanları
        b. Olumsuz yanları nelerdi?
    d) Seçme şansınız olsaydı geleneksel tip programlama eğitimini mi tercih ederdiniz yoksa görsel programlama eğitimini mi? Neden?
2. Bu dönem aldığınız visual programming dersinden ve derste yaşadığınız deneyimlerden bahsedebilir misiniz?

    a. Yapılan etkinliklerden keyif alıyor muydunuz?
    b. Yapılan etkinliklerden sıkıldığınız yönler nelerdi?
    c. Ders haricinde bu derse ne kadar zaman ayırıyordunuz?
        i. Ödevler ve proje için?
        ii. Anlamak veya kendinizi geliştirmek için?

3. Sizce visual programming dersinin diğer dersler için yarar sağlayabilir mi
4. Bu derste edindiğiniz algoritma bilgileri gündelik hayatınızda işinize yaradı mı/yarar mı?
5. Derste eksik gördüğünüz noktalar var mı? Nelerdir
6. Dersin sevdiğiniz yönleri nelerdir?
7. Öğrenmekte güçlük çektiğiniz veya yardıma ihtiyaç duyduğunuz noktalar var mı? Varsa
    a. Hangi konularda sıkıntı yaşadınız?
    b. Bu sorunları çözmek için ne gibi öneriler sunabilirsiniz?
8. App Inventor ortamının size kattığı şeyler nelerdir?
9. Ortamın olumlu yanları nelerdir?
10. Ortamın olumsuz yanları nelerdir?
11. Ders dışında uygulama geliştirmeyi düşünüyor musunuz?
12. Ders bittikten sonra
    a. App inventorla başka uygulamalar geliştirmeyi veya profesyonel yaşamınızda kullanmayı düşünüyor musunuz?
    b. Ders kapsamında veya gönüllü olarak üst düzey bir programlama dili öğrenmeyi düşünüyor musuz?
    c. Android programlamada ilerlemek daha ileri düzey bir ders açılırsa almak ister miydiniz?

APPENDIX B: APPROVAL OF ETHICAL COMMITTEE



UYGULAMALI ETİK ARAŞTIRMA MERKEZİ
APPLIED ETHICS RESEARCH CENTER

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 22 91
F: +90 312 210 79 59
ueam@metu.edu.tr
www.ueam.metu.edu.tr

Sayı: 28620816/

12.03.2014

Gönderilen : Prof. Dr. Kürşat Çağıltay
Bilgisayar ve Öğretim Teknolojileri Eğitimi

Gönderen : Prof. Dr. Canan Özgen
IAK Başkanı

Canan Özgen

İlgi : Etik Onayı

Danışmanlığını yapmış olduğunuz Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü öğrencisi Kadir Yücel Kaya'nın "Developing an Instructional Theory for Novice Programming Students to Teach Visual Programming" isimli araştırması "İnsan Araştırmaları Komitesi'' tarafından uygun görülerek gerekli onay verilmiştir.

Bilgilerinize saygılarımla sunarım.

Etik Komite Onayı

Uygundur

12/03/2014

Canan Özgen

Prof.Dr. Canan Özgen
Uygulamalı Etik Araştırma Merkezi
(UEAM) Başkanı
ODTÜ 06531 ANKARA

APPENDIX C: INFORMED CONSENT FORM

GÖNÜLÜ KATILIM FORMU

Bu çalışma ODTÜ Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü bünyesinde doktora öğrencisi Arş. Gör. Kadir Yücel Kaya tarafından Prof. Dr. Kürşat Çağıltay gözetiminde yapılmaktadır. Çalışmanın amacı başlangıç düzeyindeki programlama öğrencileri için etkili, verimli ve motive edici bir öğretim teorisi tasarlamaktır.

Çalışma süresince video kamera ile kayıt altına alınacaksınız. Bu kayıtlar da dâhil olmak üzere çalışma süresince elde edilen bilgiler ve gözlemler sadece bilimsel yayınlarda kullanılacaktır. Herhangi bir kişisel bilgi paylaşılmayacaktır.

Kullanılan programlama ortamı sürükle-bırak işlemleri ile Andorid işletim sistemine sahip telefon ve tabletler için uygulama geliştirme imkanı sağlanmaktadır. Çalışma boyunca görüşleriniz geliştirilecek öğretim teorisini şekillendirmeye yarayacaktır. Bu çalışmaya katıldığınız/katılıma izin verdiğiniz için şimdiden teşekkür ederiz. Çalışma hakkında daha fazla bilgi almak için Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü öğretim üyelerinden Prof. Dr. Kürşat Çağıltay ile (Tel:312 210 3683; E-posta: kursat@metu.edu.tr) ya da araştırma görevlisi Kadir Yücel Kaya (Tel:312 210 7519; E-posta: kykaya@metu.edu.tr) ile iletişim kurabilirsiniz.

Bu çalışmaya tamamen gönüllü olarak katılıyorum ve istediğim zaman yarıda kesip çıkabileceğimi biliyorum. Verdiğim bilgilerin bilimsel amaçlı yayımlarda kullanılmasını kabul ediyorum. (Formu doldurup imzaladıktan sonra uygulayıcıya geri veriniz).

İsim Soyad	Tarih	İmza
	----/----/-----	

APPENDIX D: CODING TABLE

Theme	**Sub-theme**	**Code**
Communication		Communication with the instructor
	Positive sides of the communication medium	Popular and common use
	Positive sides of the communication medium	Multi-directional, interactive, and open communication
	Positive sides of the communication medium	Communication medium as a resourcehub
	Negative sides of the communication medium	Direct messaging
	Negative sides of the communication medium	Using one communication medium
Contributions of the course	Transfer/link to proffessional life	Teaching programming
	Transfer/link to proffessional life	Supporting the Career
	Transfer/link to proffessional life	Need/leisure time/entertainment
		Using AI for other courses
	Computational thinking	Definition/characteristics of CT
	Computational thinking	Real-life examples/realization
		Learned concepts of programming
Motivation		Visual environment

		Non-intimidating course design
		Self-improvement
		Preferring practice to theory
		Proofs of motivation/competence
Programming and programming environment		Attitude change towards programming
	Visual only	Syntax related problmes
	Visual only	Immediate feedback and user-friendly environment
	Visual only	Concrete products
		Combination of both
	Textual only	More flexible and advanced
	Positive aspects of AI	AI is Visual / Simple / Productive
	Positive aspects of AI	Instant feedback and testing
	Negative aspects of AI	Not flexible enough for advanced programming
	Negative aspects of AI	Technical deficiencies
	Negative aspects of AI	Visual flexibility problem
Dynamics and evaluation of the course	Challenging concepts of programming	Variables
	Challenging concepts of programming	Clock
	Challenging concepts of programming	Database

	Discovery learning – reshaping the tutorials	Fewer images in tutorials
	Discovery learning – reshaping the tutorials	The more practice, the better
	Discovery learning – reshaping the tutorials	Removal of tutorials after two weeks
	Discovery learning – reshaping the tutorials	Dynamic tutorials and develop-it-more activities
	Discovery learning – reshaping the tutorials	Homework
		Blending top-down and bottom-up approach
	Theoretical hour	Verbal explanation / recitation hour
	Theoretical hour	Peer support and Idea-pitching
		Supportive materials

APPENDIX E: SYLLABUS EXAMPLE

Middle East Technical University
Spring 2014
CEIT 440 (Visual Programming)

Instructor:	Prof. Dr. Kürşat Çağıltay
E-mail:	kursat@metu.edu.tr
Teaching Assistant:	Res. Ast. Kadir Yücel Kaya
E-mail:	kykaya@metu.edu.tr
Office:	C-104
Course Hours:	Wednesday 9:40-11:30 (Lab session – in CEIT-Lab 3) Friday 9:40-11:30* (Theoretical session- in CEIT-Seminar Room) *Time and day of session can be changed based on the free hours of students

Course Description

This course is designed to provide students basics of algorithms and programming through a visual programming language. Students will develop their own android applications based on their needs. App Inventor environment will be mainly used as a visual programming environment. At the end of the course, it is aimed to teach basic concepts of programming. It is also aimed to reach a working end-product prepared by the students based on their needs.

Target Group

- Students with basic computer skills
- Students with no prior programming experience (or basic level)
- Students who are interested in application development

Resources

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). App inventor: create your own Android apps Sebastopol, CA: O'Reilly.

http://appinventor.mit.edu/explore/

http://www.appinventor.org/

http://www.youtube.com/ -> Search App Inventor

Course Website: http://ocw.metu.edu.tr/course/view.php?id=232

Grading

In-class assignments (%20) Homework (%30) Presentation and End-Product (% 50)

Tentative Course Calendar

Date	Topic
Feb 25th	First Meeting & Introduction
Mar 2-3rd	Getting to know the environment
Mar 9-10th	Using Sprites and Creating Animations
Mar 16-17th	Using Variables
Mar 23-24th	If…Else Structure and Clock Use
Mar 30-31st	Functions
Apr 6-7th	Loops
Apr 13-14th	Using Multiple Screens
Apr 20-21st	Database and Lists
Apr 27-28th	Working With Sensors
May 4-18th	Project Discussions and Development
Final Week	Project Presentations

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Kaya, Kadir Yucel
Nationality: Turkish (TC)
Date and Place of Birth: 25 November 1984, Mersin
Marital Status: Married
Phone: +90 366 280 3419
Fax: +90 366 212 3353
email: kadirkaya@kastamonu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
BS	Dokuz Eylul University CEIT	2005
High School	İçel Anadolu High School	2002

WORK EXPERIENCE

Year	Place	Enrollment
2016-Present	Kastamonu University - CEIT	Research Assistant
2009-2016	Middle East Technical University - CEIT	

FOREIGN LANGUAGES

Advanced English

PUBLICATIONS

1. Tisoglu, S., Kaya, K. Y., & Cagiltay, K. (2018). Review of Village Institutions from the Aspect of Instructional Technology. *Mersin Üniversitesi Eğitim Fakültesi Dergisi 14* (1), 463-480.
2. Tisoglu, S., & Kaya, K. Y. (2017). Proceedings from 26th EDEN Annual Conference: *Exploring the Use and Creation of a MOOC Environment: A Case Study*.
3. Kaya, K. Y., Gulec, M., Esfer, S., Tisoglu, S. & Kara, E. (2017). Proceedings from 26th EDEN Annual Conference*: Research Trends of Instructional Technology Dissertations in Turkey*.
4. Kaya, K. Y., & Cagiltay, K. (2017). Creating and Evaluating a Visual Programming Course based on Experience of Students. In Rich, P., & Hodges, C.

(Eds.), *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 135-151). Springer International Publishing
5. Kaya, K. Y., Gulec, M., & Cagiltay, K. (2016). Proceedings from ICITS 2016: Examining Visual Programming Products of Novice Programmers: A Case Study.
6. Kara, E. Eşfer, S., Güleç, M., Tısoğlu, S., Kaya, K. Y. (2016). Proceedings from ICITS 2016: *2003-2015 yılları arasında Öğretim Teknolojileri Alanında Yazılmış Doktor Tezlerinin İçerik Analizi*.
7. Tisoglu, S., Kaya, K. Y., & Cagıltay, K. (2016). Proceedings form OE Global 2016: *Exploring the Non-obligatory Use of Open Educational Resources: A Case Study*
8. Kaya, K. Y., Tisoglu, S, Cicek, M. (2015). Proceedings from AECT 2015: *Creating and Evaluating a Visual Programming Course based on Experience of Students: A Case Study*.
9. Cicek, M., Kaya, K. Y., Tisoglu, S. (2013). Proceedings from AECT 2013: *The Reasons of Playing/Not Playing Games on Facebook: University Students Case.*
10. Tisoglu, S., Kaya K. Y., Cagiltay, K. (2013). Proceedings from ICITS 2013: *Examining the Village Institutions from the Aspect of Instructional Technology: Literature Review*.
11. Kaya, K. Y., Tisoglu, S., Ucak, S. S. K., Kadioglu, E. A. (2012). Proceedings from EDULEARN'12: *Perceptions of Prospective Information Technologies Teachers towards Fatih Project and Its Components.*
12. Kaya, K. Y., Tisoglu, S., Ucak, S. S. K., Kadioglu, E. A. (2012). Proceedings from ICITS 2012: *Investigating Technological Components of Fatih Project: A Review Of Literature.*
13. Uzun, C., Kaya, K. Y., Kursun, E., & Cagiltay, K. (2011). Proceedings from ICITS 2011: *Critical Points and Dynamics of Instructional Design and Development Process in the Creation of Learning Material for Teaching Basic Concepts to Students with Mental Disabilities via Multitouch Screen.*
14. Kaya, K. Y. & Cagiltay K. (2011). Proceedings from ICITS 2011: *Perceptions Of Turkish Computer Education And Instructional Technology Program Students and Alumni Towards Game Use in Education.*

HOBBIES

Video games, Computer Technologies, Movies, Literature