12-31-2021

Improving programming learners' experience through interactive computer tutor based MOOCs

Ruiqi Shen
New Jersey Institute of Technology, rs858@njit.edu

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

IMPROVING PROGRAMMING LEARNERS' EXPERIENCE THROUGH INTERACTIVE COMPUTER TUTOR BASED MOOCS

by
Ruiqi Shen

With the large demand for technology workers all around the world, more people are learning programming. Studies show that human tutoring is the most effective way to learn for novice programmers. However, problems such as the inaccessibility to physical classes, prohibitive costs, and the lack of educators may limit students' opportunities to learn from these resources. Additionally, because programming is a skill requiring continuous practice and immediate feedback, simply listening to lectures may not be sufficient to learn effectively. This increases the inconvenience of learners who use online learning tools such as Massive Open Online Courses (MOOCs).

In recent years, a particular type of MOOC that teaches programming in an interactive manner has become popular among programming learners, such as Codecademy and Treehouse. These systems are described as *interactive computer tutors* (ICTs) in this dissertation. ICTs provide an efficient solution for programming learners to practice the idea of learning-by-doing. The commercial application of ICTs has been growing rapidly in recent years and has gained a broad user base. Despite their success, there is limited research in the literature that addresses the users of ICTs. For example, who are the learners, what do they think of these ICT based MOOCs, and how can we improve their learning experience?

This dissertation examines how learners interact with ICT based MOOCs and what design features improve their experience. Four studies were conducted to answer these questions. The results from these studies indicate that learners' level of autonomy require different needs from ICTs, which are not addressed by current

ICT designs. In addition, an autonomous feature is tested on an interactive computer tutor that teaches programming, and the effects for different learners are examined.

IMPROVING PROGRAMMING LEARNERS' EXPERIENCE THROUGH INTERACTIVE COMPUTER TUTOR BASED MOOCS

by
Ruiqi Shen

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Information Systems

Department of Informatics

December 2021

APPROVAL PAGE

IMPROVING PROGRAMMING LEARNERS' EXPERIENCE THROUGH INTERACTIVE COMPUTER TUTOR BASED MOOCS

Ruiqi Shen

Michael J. Lee, Dissertation Advisor — Date
Associate Professor of Informatics, New Jersey Institute of Technology

---

Aritra Dasgupta, Committee Member — Date
Assistant Professor of Informatics, New Jersey Institute of Technology

---

Margarita Vinnikov, Committee Member — Date
Assistant Professor of Informatics, New Jersey Institute of Technology

---

Sang Won Lee, Committee Member — Date
Assistant Professor of Computer Science,
Virginia Polytechnic Institute and State University

Austin Z. Henley, Committee Member — Date
Assistant Professor of Electrical Engineering and Computer Science,
The University of Tennessee, Knoxville

BIOGRAPHICAL SKETCH

Author: Ruiqi Shen

Degree: Doctor of Philosophy

Date: December 2021

Undergraduate and Graduate Education:

- Doctor of Philosophy in Information Systems,
  New Jersey Institute of Technology, Newark, NJ, December, 2021
- Master in Information Systems,
  New Jersey Institute of Technology, Newark, NJ, December, 2017
- Master in Human Resource Management,
  Rutgers University, New Brunswick, NJ, May, 2015
- Bachelor of Management in General Administration,
  Zhongnan University of Economics and Law, Wuhan, China, June, 2013

Major: Information Systems

Presentations and Publications:

Michael J. Lee and **Ruiqi Shen**. Autonomy-Supportive Game Benefits Both Inexperienced and Experienced Programmers. *Journal of Computing Sciences in Colleges (CCSC)*, Volume 37, Issue 2, pages 89-97, 2021.

Ruiqi Shen and Michael J. Lee. Learners' Perspectives on Learning Programming from Interactive Computer Tutors in a MOOC. *IEEE Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1-5, 2020.

Ruiqi Shen. Interactive Computer Tutor as a Programming Educator: Improving Learners' Experiences. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1-2, 2020.

Ruiqi Shen, Joseph Chiou, and Michael J. Lee. Becoming Lifelong Learners: CS Learners' Autonomy. *Consortium for Computing Sciences in Colleges - Northeastern Region (CCSC-NE)*, pages 1-2, 2020.

Ruiqi Shen, Donghee Yvette Wohn, and Michael J. Lee. Programming Learners' Perception of Interactive Computer Tutors and Human Teachers. *International Journal of Emerging Technologies in Learning (iJET)*, Volume 15, Issue 9, pages 123-142, 2020.

Ruiqi Shen, Donghee Yvette Wohn, and Michael J. Lee. Comparison of Learning Programming Between Interactive Computer Tutors and Human Teachers. *ACM Conference on Global Computing Education*, pages 2-8, 2019.

This dissertation is dedicated to my parents Wenjun and Zhiguo, my husband Bowen, and my son George.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my dissertation advisor, Dr. Michael J. Lee, for all the help and guidance he provided throughout my doctoral program. Thank you for your advice on paper writing, experimental design, research, as well as on life and career.

I would also like to thank my dissertation committee members, Dr. Aritra Dasgupta, Dr. Margarita Vinnikov, Dr. Sang Won Lee, and Dr. Austin Z. Henley, for their time to participate in my dissertation, and their insightful comments and questions. It is my great honor to have their guidance and help to complete this dissertation.

I am very thankful to Dr. Brook Wu, Dr. Starr Roxanne Hiltz, Dr. Donghee Yvette Wohn, and Dr. Shaohua Wang for all the advice and encouragement they have given me. Special thanks are given to our department assistants, Ms. Patricia B. Lundberg and Ms. Jacinta Williams, for all their great assistance.

I would also like to thank the Department of Informatics and the National Science Foundation (DRL-1837489 and IIS-1657160) for funding my research.

Also, a special thanks to all the colleagues and friends, Dr. Yu Xu, Dr. Yang Liu, Jie Cai, Dr. Mashael Almoqbel, Dr. Han Hu, Dr. Ye Xiong, Mr. Joseph Chiou, and Dr. Eric Nersesian. For years, they have inspired my research ideas, given me practical help, and encouraged me when I felt discouraged.

Finally, I give thanks to my mother Wenjun You, father Zhiguo Shen, son George Pan, and beloved husband Bowen Pan. I could not have done this work without their understanding, sacrifice, and unconditional love.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Programming is an ability that learners develop over time, allowing them to apply knowledge effectively and readily to solve problems [177]. Therefore, learning programming is largely a learning-by-doing effort that requires prompt feedback and many problem-solving practices [5] for learners to make progress [96].

One-on-one tutoring is one of the most effective ways to gain sufficient practice and receive prompt feedback [79]. However, the lack of computing teachers continues to be a concern of researchers and educators [80]. Moreover, many learners have limited access to in-person and personalized programming courses. Even for those who have access to courses, a large lecture-based format is not an ideal setting for teachers to pay attention to the individual needs of each of their students [10].

With the development of new technologies for communication and education, many learners are turning to new media such as massive open online courses (MOOC) to gain cheap and quick access to educational topics of interest. For learning programming specially, there is one type of MOOC that provides courses in an interactive manner which has gained it popularity in recent years.

One such popular MOOC is *codecademy.com* [164] (see Figure 1.1), a website that provides interactive programming courses. Learners can read the textual instruction while typing code within the same window. They can also get instant feedback for the code output and exercises. Anderson et al. [9] were among the first to introduce this kind of system to teach programming, describing a two-component system: a "problem solver" (which can interpret learners' code and provide instant feedback) and an "advisor" (which provides guidance to learners throughout the

learning process). We use this description to define interactive computer tutors (ICTs) in this dissertation.

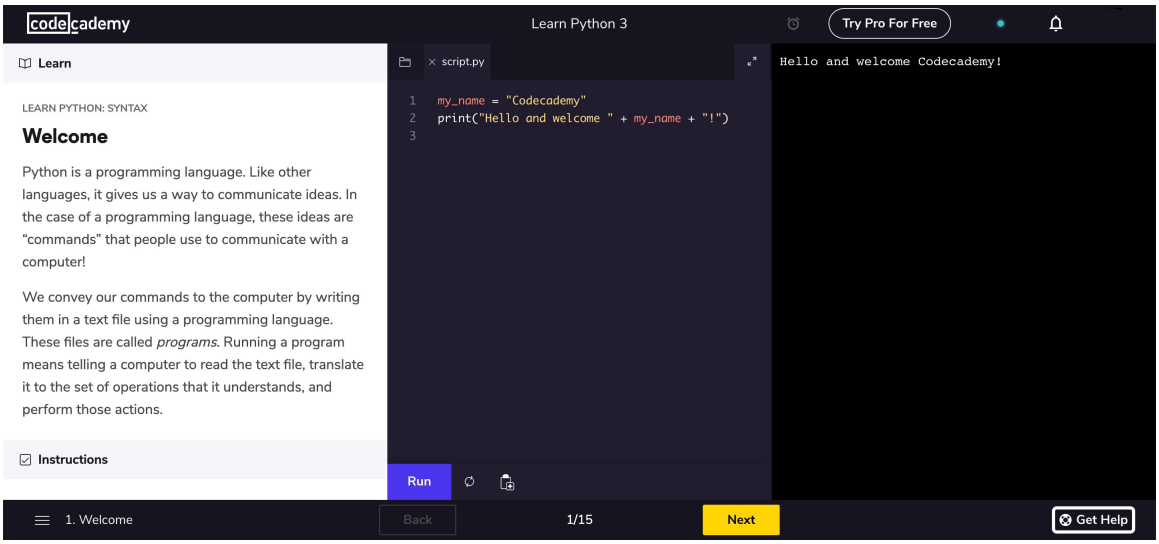

Figure 1.1 The interface of Codecademy.

Despite its popularity, few studies have examined these ICT based MOOCs, especially from the learners' perspective. We believe that it is important to understand the learners, since it points out a direction of system design that could improve learners' experience, and furthermore, increase their engagement and potentially improve learning outcomes. Based on this observation, we set out to investigate ICT based MOOCs using the Human-Computer Interaction (HCI) approach, emphasizing the users' (or more specifically in this case, learners') interactions, experiences, and needs with such a system. With this initiative, we conducted four studies.

The results from our four studies suggest that programming beginners are the main users of these kinds of systems, since these systems are good at delivering basic programming concepts and providing intensive practice. Experienced learners also use these systems to learn new languages and refresh their existing skills. Beginners have lower level of autonomy—meaning that they have less control over their learning process—and therefore, they prefer a well-structured curriculum and a system that

can force them to learn step-by-step. While for more experienced learners, they tend to have higher level of autonomy, and so they prefer controlling their own learning process by enjoying the freedom of moving about freely within a learning environment.We also found that an autonomous feature would affect both.

The contributions of this dissertation are:

1. Among the first to explore ICT based MOOCs from the learners' perspective;
2. Providing a unique view to classify programming learners, which is from the level of learn autonomy;
3. Identifying autonomous needs of different programming learners;
4. Applying different autonomous settings to improve learners' learning experience;
5. Contributing to the field of human-computer interaction, computing education, and learning science that would be useful to users, researchers, and designers of interactive learning systems for programming education.

1.2 Research Objectives

This research has two main objectives, the first is to understand the learners of ICT based MOOC, which includes: 1) who the learners are, 2) why they choose it to learn, and 3) what they think of it. The second objective is to explore how to improve learners' experience with ICT based MOOC. The second objective was built upon the understanding of the first one.

These research objectives are framed in the scope of programming education. Since ICT based MOOC has been widely used for learning programming, therefore, ICT based MOOC in this dissertation refers to an interactive MOOC that teaches programming courses, and learner refers to programming learner.

To reach the objectives, the methods used in this dissertation include qualitative methods such as content analysis and interviews, and quantitative method such as questionnaire and database collection.

1.3 Dissertation Organization

This chapter provides a brief introduction and background of our work in general, in the following chapters, the detailed background, theoretical framework, studies and findings will be provided.

Chapter 2 summarizes related work, particularly about systems for learning programming from ICTs. This chapter also specifically defines how the term "ICT" will be used throughout the document, outlines the theoretical framework and practical application of interactive learning, and further discusses the state-of-the-art in ICT research.

Chapter 3 details our first study, in which a content analysis was conducted. The goal was to understand the users (mostly the learners) of Codecademy. Reviews about Codecademy on Quora (a well-known Q&A website) were extracted to be used for inductive analysis. In this study, results indicate that 1) Beginners were the main users, while experienced learners also used it to learn new languages or refresh skills, 2) it was good at delivering web development/front end courses, 3) interactive environment increased the engagement of learners and 4) learners complained that the courses are not practical. The implication for HCI research and for next study were also discussed in this chapter.

Chapter 4 explains our second study, in which 20 programming learners were interviewed about their experience with and preference between ICTs and teachers. Learners' perceptions for both learning methods were found and discussed, including their likes and dislikes. Some patterns were also found, such as 1) experienced learners had higher autonomy than less experienced ones, and 2) learners with higher autonomy would prefer learning from autonomy supportive system, and vice versa. The implications for both ICT designers and CS educators were discussed, and the patterns we found inspired next study.

Chapter 5 presents our third study, in which CS learners' autonomy were investigated. The theoretical framework of learner autonomy were provided, including the definition, manifestation and research in various educational context. A questionnaire was distributed to 364 CS learners, scaling their autonomy level and system preference. The results indicated that: 1) CS learners had an overall medium to high levels of learner autonomy, 2) Experienced CS learners tended to have higher learner autonomy than beginners, 3) CS learners preferred using autonomy-supportive systems to learn, and 4) learners with higher autonomy would prefer autonomy-supportive system more than those with lower autonomy.

Chapter 6 describes our fourth study, in which an autonomous feature was added in an ICT based debugging game. The effect of this feature on learners with different levels of autonomy was examined. We found that: 1) learners with higher autonomy used the autonomous feature more, and 2) learners with lower autonomy completed more levels.

Chapter 7 concludes the whole work, discusses the contribution, and proposes future direction for this research.

The diagram shown in Figure (1.2) depicts the connection among each study, and the connection between the studies and the overarching goal.

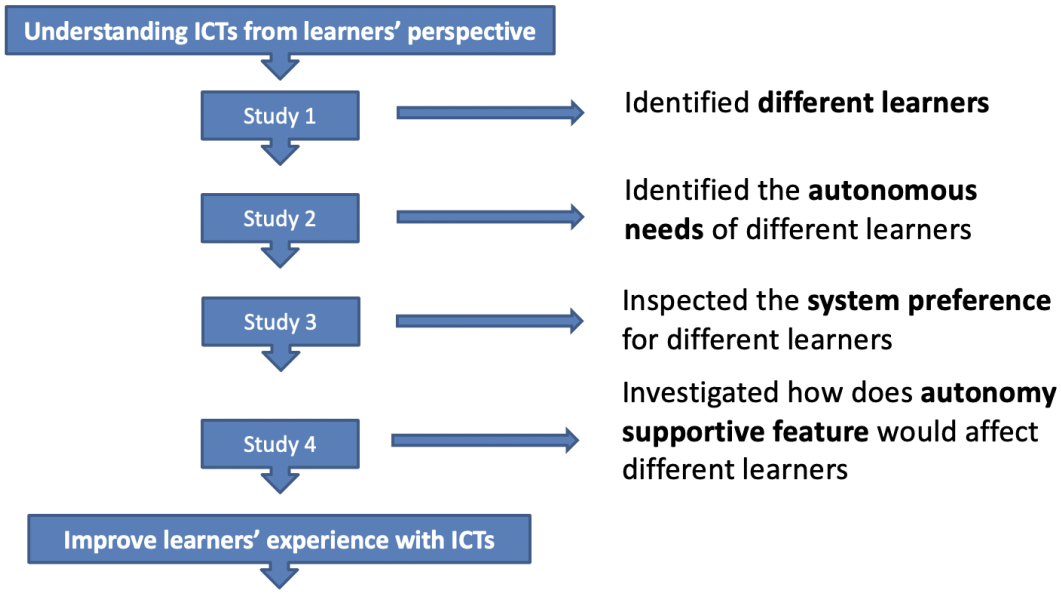

Figure 1.2 The connection between studies and the overarching goal.

CHAPTER 2

LEARNING FROM INTERACTIVE COMPUTER TUTORS

2.1 Introduction

This chapter will first discuss interactive learning theory and practice, and then define the most important term in this dissertation, the *interactive computer tutor* (ICT). Related work of the application of ICT in programming education will be discussed to highlight the state-of-the-art in the field, while also revealing the knowledge gap in current research.

2.2 Interactive Learning System in Theory

In the view of constructivism, learners construct knowledge by building on what they already know [174] (i.e., learning-by-doing). Therefore, educators ascribing to this learning theory encourage and provide their students with experiences that help them build on their current knowledge of the world [59]. From this point of view, learning is not a one-way flow of information from teacher to students, but rather an interactive process. Learning interactively is fundamental and important for knowledge acquisition and skill development [18]. Therefore, increasing meaningful interaction in pedagogy and improving its quality have long been goals for researchers and educational system designers [6, 8, 82, 170]. To enhance the interactive learning practice, Woo & Reeves [70] recommended re-conceptualizing web-based interactive learning based on social constructivism learning theory. In their view, the use of authentic activities in interactive learning systems [85] is considered meaningful interactions [185].

From a more practical point of view, interactive learning system has been applied in different educational contexts and in various formats, and the use of it

in formal education has been shown to have positive effects on learning gains and students' motivation. Stillson & Alsup [167] used an online, interactive learning system called ALEKS to teach basic Algebra to students, and they found that students' understanding of algebra was enhanced by using this system. Similarly, Wade-Stein & Kintsch [179] applied interactive computer support for writing, which resulted in improved student engagement and learning outcomes. Another interactive system, STAR, used voice recognition technology to help children learn word pronunciation. This system improved their speech skills and motivation to learn [152]. Lee et al. [112] used an interactive e-learning system using pattern recognition and augmented reality, finding that the system was helpful in engaging elementary students to learn school curricula.

To summarize, many studies have used interactive learning system to teach entry level courses and basic skills, and demonstrated that using it could improve learner's engagement. However, there are limited views on the application of interactive learning system in more advanced learning scenarios.

2.3 Define Interactive Computer Tutor

According to Anderson's ACT-R theory [8], the acquisition of cognitive skills points out a direction for instruction method, in which students are presented with both declarative instruction and a series of guided practices [9, 8, 11]. This theory is the foundation for several instructional design practices, including programming education. An application of it is called Cognitive Tutor, which Anderson et al. used to build an interactive computer tutor that teaches LISP programming language to students [9, 8]. In their work, they described the tutor, which is called GRAPE (Goal Restricted Production Systems), a two-component system: a "problem solver" (which can interpret learners' code and provide instant feedback) and an "advisor" (which provides guidance to learners throughout the learning process). In addition, this

system contains both tutorials and coding environment in one window, and the system interacts with students by providing guidance, hints, queries and feedback [9, 60]. Based on their description, this dissertation adopts their approach and defines *interactive computer tutor* as a system that provides both declarative knowledge and problem-solving practices, and that guides through the learning process by interacting with students.

Any system that corresponds this description will be included in the scope of ICT defined in this thesis. A typical example is Codecademy (see Figure 1.1).

To further define the scope of ICT for programming education, here is a list of features that are considered should not be included in the scope. 1) a system that only provides tutorials, either in the form of text or video, but does not have any interactions with learners; 2) a system that has interactive activities, but does not provide a code editor; 3) a system that has tutorials, interactive activities, and code editors, but does not provide interactive guidance during the problem solving process, e.g., no feedback or instruction when learners are doing the coding practices.

2.4 Interactive Computer Tutors in Programming Education

Programming is the ability that learners develop to apply knowledge effectively and readily to solve problems using computer code [177]. Therefore, learning programming is largely a learning-by-doing effort that requires many problem-solving practices [5], and immediate feedback is important for learners to make progress [96].

Researchers have been studying how interactivity can be increasingly better integrated with the idea of learning-by-doing and how well these methods can promote programming education. One research direction is about blending ICTs into traditional classrooms. Learning programming fundamentals has been documented to be a difficult task for students [25], mainly because of poor teaching methodologies, low interaction levels with students, and lack of students' interest [17, 46]. A number

of studies have investigated the effect of introducing ICTs into traditional classes to teach basic concepts with the goal of increasing student motivation. For example, Law et al. [103] implemented an interactive learning system—Programming Assignment aSsessment System (PASS)—which provided supportive features to students learning programming. They found interactive factors such as 'clear direction' and 'reward and recognition' can enhance learning motivation and self-efficacy. Brusilovsky et al. [30] applied ICT in formal classroom settings to teach SQL programming, showing that complementing teachers' lessons with ICT helps to increase learners' motivation. Biju [25] used Alice, an open source ICT to teach fundamentals in an introductory programming courses. The results indicated that students gained higher success rate in understanding programming concepts, and their motivation to complete assignments improved.

As for novices and children who first learn programming, Lee et al. [108] designed an interactive debugging game using ICT features that was found to effectively engage a wide range of users in learning introductory programming concepts. Similarly, Nersesian et al. [135] created an interactive AR system called CSpresso to teach middle school students how to count in binary, and found that students using the system were highly engaged, and performed as well on a post-exam as those who were taught the same concepts by a teacher.

While the aformentioned studies investigated systems focusing on learning outcomes, there are some studies that examine how students perceive ICTs. For example, Karnalim & Ayub [91] applied Python Tutor [78], a web-based interactive system that teaches python interactively by visualizing the program execution process, on programming laboratory session and collected the students' perspectives. They found that the positive comments towards the system outweighed the negative ones. One noteworthy point is that the students appreciated the system's help in finding errors for them. Similarly, Sharp [159] used Codecademy's interactive lessons as

an instructional supplement in a Python programming course, and also found that positive comments about the system greatly outweighed the negatives. From the students' perspective, while they enjoyed the interactive nature of the system, they had critiques about limited use of their creativity due to the nature of the system's predefined solutions.

To summarize, although a number of studies have shown that ICTs can increase programming learners' motivation and engagement, there are only a handful of studies that probed learners' perceptions of ICTs. A holistic picture is needed to understand learners' experiences and needs for using ICTs when learning programming to continue improving on current implementations.

2.5 Interactive Computer Tutor based MOOCs

A successful application of ICTs for programming is within ICT based MOOCs (or Massive Open Online Course), such as Codecadmy [164]. This kind of ICT has a broad user base [32], not only because they provide various programming courses [164], but also due to the (usually positive) experience provided by the integrated interactive learning environment. In these systems, the tutorial/instructions and coding panes are displayed in one application window or webpage. Learners type into the coding pane while following the tutorial, and receive feedback from the system immediately [148]. Using Codecademy as an example to illustrate the interactivity (see Figure 1.1), the interface resembles the interactive tutor described in this chapter, and proposed by Anderson et al. [9].

Recently, Kim & Ko [97] provide a general report for ICT based MOOCs, investigating 30 popular online coding sites, and found that these systems could be improved by personalized support, and more precise and contextualized feedback. One such interactive educational system is Khan Academy [158]. It applies the same kind of code editor as Codecademy does, but adds gamification elements to motivate

programming novices [176]. However, meaningful gamification in education should help learners build up engagement for long term change [136]. In learning systems such as Khan Academy, simply using a reward system may increase learners' extrinsic motivation for the short term, but may reduce their intrinsic motivation for the long term [130, 136]. This was demonstrated by Morrison & DiSalvo [130], who found that some of the gamification elements in Khan Academy were not meaningful to users. Because this study exclusively focused on novices, it raises the opportunity to learn more about non-novices, their motivations to learn, and their views about different interaction elements within these ICT based MOOCs.

Another application of ICT based MOOC is SoloLearn, a tool that learners can use in their web browser or mobile phone to learn programming. Deshmukh & Raisinghani [54] combined this application with the use of the *Application Based Collaborative* approach (working on a collaborative project) to teach students programming. This approach assumes that the interactive learning system can teach learners basic concepts, while the collaborative project can keep experienced learners engaged. The study found that this combintion of approaches positively affected learners' motivation. However, since this approach used collaborative projects to engage experienced learners, it is unclear whether ICT based MOOCs could also attract and engage experienced learners. We plan to explore this, examining how we can improve systems to serve both beginners and experienced learners based on their unique needs.

In this section, we looked at several ICT based MOOCs that are commercially available online. Although we ground our study on the users of these systems, we do not base our work on a single system. Instead, we started the work by first exploring ICTs in theory and in academic practice, we found out that although these studies have proposed several designs and pedagogy to engage learners and improve learning outcome, there lacks the learners' view about ICTs. Especially

when we look at the commercial application as next step, which is the ICT based MOOC, it has a broad learner base, however, little work has been done to study these learners. Understanding the learners is important, since it points out a direction of system design that would improve learners' experience, and furthermore, increase engagement.

The contribution of this study is beyond an analysis of a specific application or tool, but a broader perspective from the learners' point of view to improve their experience with ICT based MOOCs in general.

2.6 Summary

This chapter discussed the theoretical framework and practical application of interactive learning, and investigated the current state of ICTs in the context of programming education. While much work has been done to increase motivation and engagement for beginners through interactive learning, there does not appear to be much research done examining more advanced learners. In addition, the current literature focuses on the positive aspects and resulting outcomes of using ICTs for students, but do not explore ICTs from the perspective of the learners themselves. This key aspect—how learners perceive ICTs—is the gap in knowledge that this dissertation seeks to address.

CHAPTER 3

EXPLORING LEARNERS' VIEWS ON ICT BASED MOOCS

3.1 Introduction

The demand for technology workers continues to rise around the world. Nowadays, many people are becoming self-taught programmers, learning how to code on their own using books and online resources. Even for those who take a more formal route of learning computer science (CS) in school, learning programming outside of the classroom is common. Recently, Stackoverflow, an online community for programmers, surveyed its users and found that 37.6 percent of respondents did not have a computer science degree, but 85.5 percent of them learned a new programming language, framework, or tool without taking a formal course [165]. Prior work by Scaffidi [155] estimates that for every one trained professional programmer in the US, there were four non-professionals without any formal training, programming in their everyday jobs. Supporting these large numbers of self-taught programmers is the increasing availability of Massive Open Online Courses (MOOCs) [143]. As an alternative to physical, in-person courses to learn programming, MOOCs provide a cheaper (often free) and more accessible means to learn programming, while serving a large scale of learners [171].

As discussed in Chapter 2, Codecademy is one of the most commonly used ICT based MOOCs. Although some studies have investigated the effect of integrating Codecademy into traditional classes [61, 110, 139, 159], we lack the input and perspective from the larger population of self-taught programmers who choose to learn from ICTs. Despite their enormous number of users, we do not clearly understand what programming learners think about these ICT based MOOCs—such as Codecademy—they are using. Who are the users? Why do they choose

Codecademy and what do they think about it? Answering these questions will help us understand the importance of ICTs as a programming instructor will allow us to rethink and improve on the design features that will improve the learning experience.

Since we are among the first to probe self-taught learners' views about learning coding through Codecademy, we decided to adopt a grounded theory approach to our analysis [168]. By reviewing 218 Codecademy related answers from Quora, we extracted 3 themes and 14 codes that answer three questions: 1) who are Codecademy users?; 2) why do they choose Codecademy to learn coding?; and 3) do they have any issues using it? We found that 1) it is mainly designed for beginners learning a new language, 2) it is good at delivering programming skills that can be visualized, such as web development skills, 3) interactive environment increases the learners' engagement and 4) the biggest complaint is that the courses are not practical. The findings suggest that ICTs can be an effective tool to teach programming from beginning, especially when there is a shortage in CS educators [80]. Our contributions are: 1) the first to study the users of ICT based MOOC; 2) pointing out a direction for researchers to study how users interact with ICTs; 3) proposing design features that can improve learning experience with ICTs.

3.2 Background

The rapid growth of Codecademy raises the interest of researchers and educators. There are several studies that investigated the application of Codecademy in programming education. Lee & Ko [110] compared the learning gains for novice programmers with Codecademy and an online coding game called Gidget [109], they found that both learning tools can teach novice programmers effectively. Figueroa & Amoloza [61] studied how interactive platforms can help non-computer science learners reduce programming anxiety. They found that using a system such as Codecademy can lead to "a significant decrease in learning anxiety and an increase in

perceived learning". A more recent study introduced Codecademy as an instructional supplement in an introductory Python programming course. From an educator's view, the result was positive, indicating that ICTs can be a useful supplemental tool for formal classroom instruction [159]. Although they pointed out the negatives of using Codecademy interactive lessons, such as limited opportunity for creativity and critical thinking, this study lacks the views from the students. Olsson & Mozelius [139] explored the design of online learning environment from the learners' perspective. In their case study, they utilized Codecademy to teach a group of self-learners, and found that the learners' overall experience with Codecademy was positive, and that the immediate feedback the site provided was brought up by the users as a particularly useful feature. However, despite the study's focus on the learners' perspective when using ICT based MOOCs, there is still a gap in knowledge about how general learners perceive these systems and why they choose them to learn programming.

3.3 Methodology

Since there is limited research examining learners' perspectives of ICTs, we adopted a grounded theory methodology to explore this space [168]. For the study described in this chapter, we followed the most recent grounded theory framework suggested by Tie et al. [41], which includes six steps: purposive sampling [7], generating/collecting data, initial coding, intermediate coding, advanced coding, and the grounded theory.

We collected data from Quora (www.quora.com), a popular question-and-answer website where users post questions and others respond, either factually or in the form of opinions [141]. We chose to use Quora as our data source because: 1) it has over 50,000 users subscribed to the topic "Codecademy", which represented a large number of Codecademy users around the world; 2) the respondents of most Codecademy-related questions have their real names and background information

displayed, and 3) responses are typically high-quality and authoritative, building on a reputation system among users [144, 181].

The data collection and analysis was conducted at the same time, implementing the following steps: 1) searched for the keyword "Codecademy" on Quora's homepage, the website returned the results ranked by popularity, 2) starting from the most popular question, looked at each retrieved question, decided whether it is relevant to our research, 3) once considered relevant, we went to the question page to review the answers, 4) we extracted relevant answers to our data analysis tool called MAXQDA, 5) codes were assigned to snippets of the text using the inductive analysis approach [43] (a research method that allows findings to emerge from the themes inherent in raw text data [173]). These steps were completed iteratively until we reached the data saturation [154], meaning that there were no more new codes emerging. The data saturation was reached at around 140 answers (57 questions). To ensure reliability, we continued to retrieve and analyze 68 more answers. One researcher did the initial data collection and analysis, and another researcher verified the data and codes. Our relevance criteria required a question to be learner-focused (e.g. "Is Codecademy an effective way to learn how to program") rather than company/application focused (e.g. "How will Codecademy monetize?"). Another selection criterion was for the answer to be informational enough to extract codes. For example, a response such as "Codecademy is good" would not be selected, because words such as "good" and"bad" is too general and does not point to a specific feature or audience. In contrast, a statement such as "Codecademy is good for beginners." would be selected because it gives a specific reason to why Codecademy was regarded as good.

3.4 Findings

In total, we analyzed 62 questions and 218 answers (from unique individuals). Through inductive analysis, we consolidated this data down to 3 themes and 14 codes. Between two researchers, we reached an intercoder reliability of 0.89 and intercoder agreement of 0.93 [33]. In the following subsections, we describe our results using representative excerpts from our Quora questions and answers about Codecademy. Although we had access to the Quora users' real names and additional background information through their online profiles, we intentionally refer to them as S1-S218 throughout this chapter to give them a degree of anonymity.

3.4.1 Who are the users?

We identified three types of users of Codecademy: 1) self-taught learners, which include beginners (101/218) and intermediate learners (19/218), 2) teachers who use it as a supplementary tool to teach students (3/218), and 3) companies who use it to train staff (1/218).

Beginners We found that most users think of Codecademy as an ideal starting point to learn programming. Even experienced learners who expressed major criticisms about Codecademy tended to admit that it does a good job in teaching programming to beginners. P1, a web developer for many years, wrote: "*I do recommend the [Codecademy] tutorials for complete beginners. Though a few things may be misleading, the tutorials provide the best information (I have come across) for complete beginners to learn from. As you advance, you'll inevitably find other sources more useful. But initially they provide a pretty good foundation for web development newbies.*" Another example is P2, a programming beginner when he starting to use Codecademy 5 years prior, who stated: "*As an English teacher who had no technical background at all, I really enjoyed learning and writing code on that website. It was this website that helped me step into the world of code.*"

Experienced learners We also found that some experienced learners (learners who have prior experience in learning programming) use Codecademy as a tool to refresh their knowledge or learn the basics of a new language. For example, P3 wrote: "*Codeacademy tends to work well for experienced programmers who want to brush up on some syntax or want to learn a new language.*" Although this review seems more an opinion than an experience, we have P4, who is programming since 1985, shared his personal experience. He said: "*From the personal perspective, every now and then I go to those sites when I'm curious about a language I don't know.[...] I find sometimes that's the fastest way to learn a new skill. It doesn't matter how long you've been programming, you will always be a newbie at something.*"

Others As we mentioned above, besides self-taught learners, there are other users who use Codecademy as a training tool, such as teachers and companies. Although limited comments are found, we think it is noteworthy to mention because it suggests a broader usage of ICTs. For example, P5 seems to be a middle or high school teacher, he used Codecademy to arouse coding interests of his students. He commented: "*Codecademy feels more like an interactive game that you experiment with and, in general, tends to more lightly scratch the surface of general concepts/languages. For us, at our school, we used Codecademy in our intro course to get kids' feet wet with the whole process of programming.*" On the other hand, companies will use it to train their staff. P6 thought that Codecademy is good for beginners that have some coding experience. Tools like it can be effective for training teams. He shared his friend's experience: "*he really enjoyed them and found them pretty effective for introducing their team to the BASICS of new technologies that the team is not already familiar with.*"

3.4.2 Why do they use it to learn?

Learners find it helpful because it delivers basic content in a structured way, which makes learners pick up skills quickly and easily (93/218). And people thought it was engaging and one potential reason can be that the website provides interactive learning environment (35/218). We also encountered many discussions about learning a specific language or skill from it, such as web development skills (61/218) and Python (17/218).

Learning basic content in a structured way In addition to providing basic content, providing it in a structured way is the key. As for beginners, it is often difficult to determine the direction of learning and the essential parts of the material, so it is the educators' job to carefully point out the direction and select the relevant information [35]. Codecademy does a good job from this point of view, compared with searching for scattered information online. P7 compared it with Youtube videos and commented that: "*A word of caution to youtube videos: I found 2–3 videos pretty useful/interesting but the rest were extremely hard, and for beginners it seemed too much, as you have to grasp the concept/theory first then code.*" Another way to understand "structue" is that it builds up the knowledge incrementally. For example, P8 said: "*I found it had a larger repertoire of topics I could learn from. I also found it to be slightly more structured in terms of progressing from lesson to lesson, and Codecademy had some repetition and slight jumps in difficulty level between some exercises.*"

Learning quickly and easily While learning structurally may be a demand only for beginners, learning basic content quickly and easily is a need for both beginners and experienced learners. Codecademy breaks down the materials into small pieces followed with exercises, which are easy to digest and follow. P9's words can better describe this: "*Codecademy (at least for me) was much easier for me to learn the*

basics from because it has a very easy workflow and is kind of dummy proof to follow along with." Another example is from P10: "*But the Codecademy definitely works very well for beginners. The course, say, JavaScript one, is well designed in terms of the difficulty, how easy for a beginner to learn and apply, and the little test project also summaries well each chapter's major points.*" As for experienced learners, since they already know the theories of programming, when they learn a new language, they just need to pick up the basic syntax quickly. Codecademy provides a good environment for them. P11 was an experienced learner from his description. Although he had critiques about Codecademy, he often used it to learn new languages. He said: "*I use Codecademy A LOT. The last time I used it was for PHP. I needed to work on a wordpress site ASAP, and I'd never used PHP before. I sat down and finished the course in about an hour and a half. This made it much easier for me to jump right in and read the code I was working on.*"

Interactive environment increases engagement Many users mentioned that using Codecademy makes them engage more. Structured and easy-to-follow tutorial is one reason, for example, "*Codecademy is one of the best places to spur your interest in computer programming by giving you nice and simple exercises to get a hang of the syntax.*" said by P12. Interactive learning environment is another. "*Interactivity makes the entire learning process quite engaging and, for many people, less boring than just copying code from a book/tutorial. It can also provide great help in 'breaking the wall' that generally scares people who are approaching programming.*" said by P13. The gamified feature of interactivity is a possible reason for the fun. Just as P14 commented: "*The tutorials are made in a way that is interactive, conversational and has some gaming elements that keep you moving forward.You will appreciate the idea that they are really trying to help people who wants to learn to code but too afraid to take first step to do it.*" Another angle to interpret the "engagement" is that the

interactivity act as an external motivation for learners who are otherwise not learning by doing. We can get this point of view from P15: "*The best part about Codecademy, which makes it superior to other types of learning, is the interactive way in which they teach code. You are forced to write the code as you go along the lessons rather than sitting on your couch and passively reading a fat, old book.*"

Learning a specific language It is interesting to find many discussions about the application of using Codecademy to learn web development skills, such as HTML, CSS, PHP, and JavaScript. Most users who learned these skills from it gave good comments. With its interactive feature, Codecademy does a good job in delivering web development courses. One possible reason is that these skills are tangible, meaning that learners can see the results of their code by viewing the website they create through the interactive window. Instant feedback is received by learners because the system visualizes the results for them. P16 articulate this point of view in his comment: "*For early-stage learning, I preferred Codecademy for the following reasons: 1)Really quick feedback loops - I knew whether I was coding right/wrong almost immediately because the coding and preview screen were built in. 2) And I thought they should have started with HTML5 and CSS as the first part on 'Intro to Programming' as it's more tangible.*"

3.4.3 Complaints about Codecademy

While we found good feedback of Codecademy among learners, complaints followed. The codes we extracted are: not practical (49/218), not comprehensive (27/218), and rigid system (15/218).

Not practical When we collected the data by first reviewing the questions, we found that there are some learners who got confused after they finished Codecademy courses, because they did not know how to apply their skills to real work. These

questions bring out discussions that Codecademy courses are not practical. Many users complained that these courses are too basic to be applicable. However, as a website intended for all levels of learner, it is nothing wrong to provide basics as easy as possible. But programming learners need real projects to polish their skills, that is what they found missing in Codecademy. Just as P17 commented:"*It [Codecademy] is a good place to learn basic syntax. But, if you really want to be a good programmer, you need to solve real world programming problems and gain through that experience. The best way to do that is to either work on your own project, or contribute in an open source project.*" Similar to P17, P18 said: "*Codecademy is a great place to begin Python, but you'll need to progress to projects in order to develop real skills.*" Besides the lacking of real projects, not teaching problem solving is another reason for not practical. P19 thought that it emphasized too much on syntax, so that the part to think of problem solving is missing. He commented: "*I thought the UI was great and the instructions very clear. However, I felt there was too much emphasis on syntax and mechanics and not quite enough focus on learning how to think through problems. After a while it begins to feel more like wrote memorization than learning how to get stuff done with code.*" And in P20's opinion, "too helpful" can sometimes be a hurdle to learn: "*It [Codecademy] is, in my opinion, too helpful, so people don't learn how to debug/answer their own questions, which is a critical skill to be a good professional.*"

Not comprehensive Not comprehensive can be interpreted from two kinds of expectations. Some learners expect it to be a one-stop shop to learn both the basic the advanced topics, but they often find there is missing the advanced parts. For example, P21 commented: "*I will not recommend you to learn java from this site because I already completed 2 to 3 course on it and after completion I found that the tutorial are to basic.*" Even for learners who only expect to learn the basics, they often find it not comprehensive with the basic stuff. This expectation is addressed by

learners such as P22, he commented: "*However, Codecademy can sometimes ignore very important details in various programming languages, which can be a huge issue depending on how often / how vital it is to the language.*"

Rigid system Codecademy offers structured tutorials where learners have to follow the path strictly. While some learners find it helpful as we mentioned above, other learners think it is as a rather rigid feature. They have complaints that some contents are repetitive, so they get boring after a while. P23 had a comment for this: "*While i think that Codecademy is fun to 'play with', after a while it becomes pretty repetitive and you won't learn anything new.*" The learning system is also rigid and makes people "too relax", according to P24. We can tell from his answer: "*In my opinion it is too boring and relaxed. Just repeating language constructs after a template won't help you learn a language or technology since you will forget everything you've learned that way in a week.*" In the interactive learning environment, although learners can benefit from the instant feedback, the rigid format can sometimes make it hard to proceed. Learners have to type in the code exactly as the system requires to get the right answer.P25 complained about it: "*I didn't like its strict and rigid expectations of the answers. For example if it asks your to write print 'Hello World' and you wrote print 'Hello world' it will give you an error without telling you what it is.*"

3.5 Discussion

To summarize our findings, we found that: 1) the main users of Codecademy are beginners and experienced learners who want to learn a new language, 2) it is good at delivering courses that can be visualized, such as web development/front end courses, 3) interactive environment increases the engagement of learners and 4) learners have criticism that the courses are not practical. These results have implication for both ICT designers and researchers.

From the learners perspective, while most of the users are beginners, we cannot ignore the fact that experienced learners also represent a certain user type. As we mentioned above, beginners often lack the ability to select the learning materials and decide the direction. They need educators to hold they hand and guide them through [101]. However, experienced learners have higher autonomy as regard to self-learning. This brings up a question that beginners and experienced learners have different preference for ICT based MOOC designs. While beginners may enjoy the system forcing them to learn step by step, experienced learners may prefer more freedom to pick up whatever they want to learn quickly. Further experiments can be done to explore what specific design features will serve different types of learner.

According to Moser [131], learning programming is difficult because it is too abstract to relate to everyday life. This probably explains why many ICT based MOOCs start with web development courses [50, 89], because they let learners visualize their codes by displaying the web page they create through the learning window. The interactivity even make the process faster and easier. With the demand for learning more languages increases, Codecademy now provides more courses that cover almost all the mainstream programming languages. Although we know its success in teaching web development skills, there should be further research to study the application of it on other programming languages. Especially to understand how interactivity, as a main feature of ICTs, acts its role in delivering these courses. Another noteworthy point for interactivity is that, although we found that interactive environment increases the engagement, the reasons are hidden from us. We need more in-depth analysis to understand, from learner's view, why and how interactivity can motivate them.

As for the complaints, we can see the efforts that Codecademy made to minimize the learning barriers. The easy learning process is indeed attractive to learners. However, it brings out the concerns for practicability. Teaching problem solving is

as important as teaching syntax. Not only curriculum design can adjust the weight of problem solving, but also the exercises can be designed towards it. For example, debugging is common problem solving process for learning programming [40], using the advantage of interactive learning, designers can add more debugging exercises that practice learner's problem solving skills.

3.6 Limitations

As a qualitative study, our goal was to discover patterns within a population, and therefore no attempt was made to assign frequencies to words in the text data. Thus, our study does have limitations [138] from this point of view, but we propose future work that can reduce the effects of limitation.

First, there are ambiguities inherent in human language. Especially when the data is a one-time post, there is no way to request reviewers to explain their responses in the same context when they posted their answers. It is also a limitation that we cannot ask follow-up questions behind one sentence judgement. However, synchronized communication, such as interviews or focus groups can solve this problem. With the patterns identified, we can have a clear outline for interview protocols to conduct semi-structured interviews.

Second, the time span for the data gathered was 2011-2019. With the ever-changing nature of ICT based MOOCs (e.g., functionality, content, and usability), early answers may not be representative in current context. In addition, the sample number is limited compared with the larger user population of ICTs. To generalize the findings to the population and get more up-to-date information, quantitative method such as large-scale questionnaires may be used for followup studies. The results of this study can serve as a good outline for the questionnaire design.

Finally, there might exist participant bias, because the answers we collected were from those who were active enough within an online community to answer questions

on Quora. We may be missing the opinions from silent learners who are not active on this online platform. However, this bias is common in research that involves human subjects. Subjects who opt-in the research are those who tend to express their opinions and interests more actively. To mitigate the participant bias, we will use the patterns we identified to reach more general learner population, e.g., using questionnaires.

3.7 Summary

In this chapter, we used grounded theory to study the learners of Interactive Computer Tutor based MOOC. Using Codecademy as an example, we collected and analyzed relevant questions and answers from Quora iteratively. We collected 218 reviews in total, and extracted 3 themes and 14 codes from them. By interpreting these codes, we summarize our findings as: 1) Beginners are the main users, while experienced learners also use it to learn new languages or refresh skills, 2) it is good at delivering web development/front end courses, 3) interactive environment increase the engagement of learners and 4) learners complain that the courses are not practical. These findings indicate the importance of interactive computer tutor as a programming instructor that teaches programming from scratch, and imply design features that can improve the learning experience for learners with different needs.

CHAPTER 4

INTERACTIVE COMPUTER TUTORS OR HUMAN TEACHERS?

4.1 Introduction

Learning to program is considered a difficult process and requires continuous practice much like learning natural languages. But unlike natural languages that can be used in different situations of everyday life, learners typically program within the constraints of a computer screen [131]. The difficult nature of programming may increase the dropout rate in both classrooms [137, 188], and massive open online courses (MOOC) [140]. In addition, the quality of programming teachers and MOOCs can also serve as a factor affecting dropout rates.

Learning from either teachers or computer tutors may have obvious advantages and disadvantages. Individual tutoring is an ideal strategy for teaching and learning programming—human tutoring is one of the most effective ways to overcome programming obstacles [79], but the lack of computer teachers continues to be a concern of researchers and educators [80]. Many learners have limited access to in-person and personalized programming courses. Even for those who have access to courses, a large lecture-based format is not an ideal setting for teachers to pay attention to the individual needs of each of their students [10]. More recently, MOOCs become popular alternative or supplement for traditional classroom lectures, as they are often cheaper, more accessible, and can support more students simultaneously than traditional classrooms [171]. However, the limited amount of interaction with teachers and limited extrinsic motivators are major constraints and future development opportunities for MOOCs.

In this work, we only focus on MOOCs that offer instructions through virtual agents or interactive computer tutors (rather than those that primarily provide

instructions through text or video), such as Codecademy, Datacamp, and Treehouse. We choose to focus on ICT-enabled MOOCs instead of other types of MOOCs because the literature suggests that the former type of environments can provide effective programming instruction [48, 103] and are gaining more popularity with learners [130, 133]. Although learning programming from either ICTs or teachers have shown positive learning outcomes for learners, few studies explicitly examine learners' perceptions of the experience of using and comparing the two approaches. Exploring these ideas can surface important features to better design and highlight the effective techniques that learners seek when learning to program.

This work describes a qualitative, exploratory study examining learning experiences from the perspective of learners and compares their views on learning from an ICT and from a teacher in a classroom. Although we compare the pros and cons of ICTs and teachers, our goal is not to suggest one is better than the other. Instead, we aim to find ways to improve all learners' educational experience by exploring the best practices, qualities, and techniques used by teachers to apply to and improve ICTs, and vice-versa.

4.2 Background

4.2.1 Learning from interactive computer tutors

Mastering programming skills requires extensive practice and making mistakes [5]. Many MOOC websites, such as Codecademy and Khan Academy, integrate tutorials with extensive exercises using code editors with feedback systems [148]. Empirical studies show that students who learn programming interactively through well-designed computer systems can achieve good learning outcomes and higher self-efficacy [60, 103].

However, what are the good features of an ICT based MOOC for delivering educational programming content? According to Woolf & McDonald [186], an

effective tutor must deal with the fundamental problem of communication, therefore, they suggested a computer tutor to adapt a system's response to the student's level of knowledge. Early research by Reiser et al. [149] developed an intelligent tutor that teaches LISP programming. The main goal of this tutor, called GREATERP, was to structure students' problem-solving episodes and provide feedback and guidance adaptively. This tutor was demonstrated to be more effective than traditional classroom instruction. More recent work by Staubitz et al. [166] proposed five requirements for an ICT to deliver programming courses: Versatility (support multiple programming languages), Novice-Friendliness (UI catered for beginners), Scalability (support for many users), Security (secure students' submissions/assessments), and Interoperability (integrate into existing infrastructures). Pritchard & Vasiga [148] summarized that built-in coding environments are beneficial for students' continuity in learning-by-doing. While most educators will agree that a mentor is essential in the initial learning process for beginners, Liyanagunawardena et al. [115] showed that in an online course, the learners' community itself can act as a mentor and could possibly mitigate the issue of not having enough teachers for students. Users can also identify the benefits of features such as deliberate instructional design (designed instructions), learning analysis (self-reflecting information), and instant feedback [191].

One problem associated with ICT is that learners would "game the system". Baker defined gaming the system as "attempting to succeed in an educational environment by exploiting properties of the system rather than by learning the material and trying to use that knowledge to answer correctly" [16]. Some research has found "gaming the system" to be associated with poorer learning in ICTs [15, 180]. Therefore, Baker et al. [14] investigated the reasons that learners game the system, and according to the findings, they gave suggestions on ICT design, which include: 1) balancing the problem difficulties, 2) balancing the problem length, and 3) giving learners freedom to skip problems.

In this section, we discussed the good design features of ICT and the problem associated with it. Our study expands on these works, aiming to explore from learner' perspectives on whether these commercial systems (such as Codecademy and DataCamp), which can also be called ICT based MOOC, have the deign features mentioned above, and whether do they have the mechanism to alleviate the problem.

4.2.2 Learning from teachers

Unlike ICTs which have a relatively short history in education, human teachers have been a part of education for centuries. Studies have shown the effectiveness of human teachers [26, 83, 125]. A teacher can guide students and can effectively time how much thinking a student should do before providing hints or answers directly [65]. This is especially important for novices, who can benefit more from interactions with teachers [132]. Teachers can also intervene at the right time to prevent students from becoming too frustrated [125], which is especially important in the early stage of learning, when learners have a higher likelihood of quitting [188]. Robins et al. [150] concluded that an effective programming class should raise students' interest and participation by setting clear goals and actively involving participants in course materials and problem-solving activities. However, questions remain about what specific teaching methods contribute to an effective programming class. Pears et al.'s [145] overview of programming classes found little systematic evidence to support any particular teaching approach that answers this question. Tan et al. [169] conducted a survey to gain insight from the learners' perspective and discovered that programming learners found the practical application of programming to be the most difficult, and therefore considered lab sessions with consultation more helpful than lectures. However, questions such as what kind of lab sessions they like and whether they could get sufficient consultation opportunities remain largely unknown to us.

4.2.3 Comparing/Combining interactive computer tutors and human teachers

Means et al. [123] conducted an extensive literature review comparing online learning and face-to-face learning. They found that on average, students in online learning situations (including teachers teaching in an online setting) outperformed those in face-to-face situations. Merrill et al. [125] were the first to give a comprehensive comparison between the effectiveness of human tutors and computer tutors, focusing primarily on feedback. They found that feedback from both human tutors and computer tutors help students detect and fix errors and overcome obstacles. However, human tutors outperformed computer tutors in communicating the diagnosis of the errors to students, and therefore, the highly interactive nature of tutor-student communication led to better motivational benefits compared to computer-student communication. Human tutors also outperformed in encouraging students to spend more effort to solve problems. Another major difference of feedback is that human tutors can strategically moderate their intervention while most of the computer systems cannot. As a conclusion, they suggested that computer tutors could be improved by capturing the features of a human tutor in a model-tracing way, and they further encouraged more empirical research on the differences in motivational outcomes of feedback from both computer and human tutors.

For teaching programming specifically, Warren et al.[182] compared the feedback from both computer tutors and teachers in a classroom setting. They observed that though computers could give instant feedback on whether students were correct or not, they could not accurately describe where and why answers were wrong. Furthermore, computers are unable to suggest different types of coding styles the way a teacher could. Conversely, it may be difficult for teachers to give individual feedback in a timely fashion, especially when they have a large number of students with complex assignments.

Since both computers and teachers exhibit positive and negative qualities, researchers are trying to blend the two methods and prove its validity. Heffernan & Koedinger [84] proposed a computer tutor built based on observation of experienced human tutors. This intelligent tutor, called Ms. Lindquist, could not only model trace students' actions, but could also do more human-like activities such as hold conversations and provide explanations on request. Deperlioglu & Kose [52] found that a combination of computer learning and face-to-face learning achieved more effective and efficient educational experience than traditional face-to-face classes in terms of delivering programming education. Beyyoudh et al.'s [24] work also indicated that a combination of intelligent tutoring system and human tutors increased learners' motivation.

4.2.4 Learning from the learners' perspective

Based on our literature review, we found a gap in knowledge examining learners' perspectives on receiving instruction from either ICTs or teachers. It is important to examine learners' perceptions of the differences between computers and teachers, and their preferences when interacting with either of these choices when learning programming. The chosen educational guide can influence their perception of the topic and activity and therefore may affect retention rates and learning experience.

Therefore, our overarching research objective is to better understand the learners' perspective towards ICTs and human teachers, and how we can use this knowledge to improve both ICTs and human instruction in classrooms. To address this goal, we raise the following research questions which we will explore in this article:

RQ1: What do learners (a) like, and (b) dislike, about learning programming from ICTs?

RQ2: What do learners (a) like, and (b) dislike, about learning programming from teachers?

RQ3: What do learners think are the pros and cons of feedback from ICTs and teachers?

RQ4: Do learners prefer to learn programming from ICTs or teachers?

4.3 Methodology

To answer these research questions, we conducted 20 in-person, semi-structured interviews. We chose to use interviews as our means of data collection as they are a commonly used in exploratory work [151, 183], especially when there is limited research in the literature to gain an in-depth perspective into subjects' views and experiences [72]. We used a snowball sampling method to recruit our participants [74], where we asked each participant to recommend people they knew that met our inclusion criteria and that they thought would be a good candidate for us to interview. The initial six participants were students who responded to recruitment e-mails sent to mailing lists at two different public universities in the northeastern United States. Subsequent participants were classmates, alumna, or professional colleagues of these initial six participants, representing a wide range of demographics (e.g., gender, ethnicity, age, job/major), distributed across the US.

One researcher conducted all the interviews. 16 interviews were conducted in-person and 4 occurred over the phone. All interviews were audio recorded, averaging 29 minutes per interview. The inclusion criteria for participants was that they had experience learning programming from both teachers and ICTs. We defined a teacher as a human instructor in a classroom setting, and used Farrell et al.'s [9] two-component definition for ICTs. We intentionally did not constrain ICTs to certain systems any further, as we wanted our participants to talk broadly about the different technologies they had used without being limited to a specific ICT. The interview questions were divided into two main parts: 1) behavioral questions that asked participants about their occupations, majors, coding experience, and coding-related behaviors (e.g., "In general, how long have you been programming?");

and 2) research-related questions that probed participants about their experiences learning programming from both ICTs and human teachers (e.g., "What problems did you encounter, and how did you resolve them?").

All of the recorded interviews were transcribed and coded using NVivo. Two researchers conducted the coding, using the three-stage coding process outlined by Cambell et al. [33] in their work describing how to measure intercoder reliability for semi-structured interview studies. This was done iteratively until the two researchers came to a consensus on codes and a sufficient level of intercoder reliability and intercoder agreement (stages 1 and 2); then the full set of transcripts were analyzed (stage 3). Since participants could state their likes, dislikes, and preferences in every research related question, we read through all the transcriptions and assigned tags to any emergent patterns (e.g., code editors, content design, flexibility, and efficiency). After assigning the initial tags, we read through those tagged texts, and consolidated similar tags into one tag (i.e., code families), or split one tag into different tags. Finally, we identified 19 themes (each research question has several themes) and more than 50 tags. We reached a high level of intercoder reliability (.87) and intercoder agreement (.92). We note that while well-established in quantitative work, there is no community consensus about the applicability of inter-rater/coder reliability measures for qualitative studies [12, 81], so we include our scores for completeness for the analysis of 20 interview transcripts. Next, we present representative quotes from participants to better explain our themes.

4.4 Findings

Our participants included 9 females and 11 males, ranging from 22 to 32 years old (median 26). Everyone was from a STEM field/major or job, consisting of 13 students (6 females and 7 males) and 7 working professionals (3 females and 4 males). Their programming experience ranged from 1 to 15 years (median 4.5).

4.4.1 RQ1a: What do learners like about learning programming from ICTs?

Provides a code editor — a code editor that is embedded with the ICT, which allows learners to write and run their code directly within the system. There are three main reasons that made our participants consider this helpful: First, a code editor dismisses the need to set up a local programming environment. 7 out of 20 participants mentioned that they only wanted to learn some basics, and that they did not want to spend time and effort to set up a local environment. Therefore, an embedded code editor saves time and effort from having to set up a local programming environment. P3, a 26-year-old male told us: "*I think it's a powerful tool, because if we want to try something on local, you need to set up your environment, you need to spend a lot of time for those.*" (P3, 6 years of programming experience)

Second, a code editor provides one-window convenience. Participants mentioned that they liked embedded code editors because they displayed tutorials, examples, and exercises in the same browser pane, which was more convenient for them to do exercises, rather than switching windows/tabs/applications between tutorials and coding tools. P9 told us how she found embedded code editors to be convenient: "*If I follow YouTube, it's not convenient because I code on my local computer, I watch the video, then switch to my software. But in Dataquest, the screen is separated in two parts. You can see the instruction and at the same time, you can type your code.*" (P9, 3 years of programming experience)

Third, a code editor provides a similar, but better-than-real environment. Two participants mentioned that they liked the embedded code editor because it was similar to a real coding environment (e.g., a local programming environment) but better because a code editor in an ICT gives customized feedback while a real environment does not. "*If you actually coding in R, like R studio, you don't know*

whether you're doing right or wrong, but this [Codecademy] actually gives you a feedback." said P19 (P19, 3 years of programming experience).

Content design — the course materials and the way ICTs organize and deliver information, broken down into important components: outline, practice, and examples. First, ICTs have a clear outline organizing and displaying lessons. The outline allows users to see exactly where they are in the learning process (i.e., curriculum). P20 described how the organization was useful in Dataquest: "*The lessons are very simplified. They are broken down into different modules, so it makes it very easy to consume.*" (1 year of programming experience). P1 compared it with tutorials from Youtube and commented: "*like on YouTube, everything is scattered but on Codecademy, they have particular syllabus that is planned for you.*" (P1, 6 years of programming experience)

Second, opportunities for practice are provided immediately after each lesson, so learners can (re)apply what they learned into practice promptly. P3 elaborated how immediate practice was useful: "*for the W3 school, you'll first grab the same concept, but immediately you will use the 'try it yourself' demo page. You can put this knowledge into real world practice. That's why I like it.*" (P3, 6 years of programming experience)

Third, examples provided along with each lesson. Participants mentioned that the examples from ICTs were very helpful to understand lessons. As a novice programmer, P5's biggest concern was that he could not visualize his code's output. He described how examples in Codecademy helped him learn web development: "*It [Codecademy] has an example to show you the final version, you can test again and compare your code to the example, that will help you to improve your code.*" (P5, 1 year of programming experience)

Flexible — this allows learners to go at their own pace whenever and wherever they want. 8 participants mentioned that learning from ICTs satisfied their desire to learn at their own pace. P2 told us that he preferred online learning for this reason: "*For [classroom] lectures [that are] 3 hours long, if you don't understand something an hour in, then you kind of waste two hours. Whereas you can make sure you understand it online before proceeding onto the next section or the next concept.*" (P2, 4 years of programming experience).

Participants also highlighted flexibility in location, such as P20, who said: "*I don't have Python installed on my phone and I can still do my lessons [on Dataquest] even if I'm in transit traveling somewhere.*" (P20, 1 year of programming experience). In the case of P8, he was a full-time student who also held a part-time job. He told us how time flexibility helped him learn. "*I could do it at 2am if I want. A teacher is not available at 2am,*" he said, (P8, 5 years of programming experience).

Efficient — ICTs helped with learning more efficiently compared to other resources. When comparing the time spent learning from an ICT and from a teacher in a classroom, two participants felt that being present in a classroom physically was time-consuming, just as P19 told us: "*Because if I want to go to school and take a class, that's going to be very time-consuming.*" (P19, 3 years of programming experience).

When comparing ICTs with textbooks, four participants thought that learning from ICTs helped them apply skills more efficiently than reading textbooks. For example, P7 told us: "*I think for textbook resource, one annoying thing is it doesn't show you like all the command[s] and what it does. So, you have to waste time reading it yourself, but for Codecademy, they just teach you each command. It's a faster way to learn it.*" (P7, 6 years of programming experience). P15 gave us a very interesting point that she can be more concentrative on computer tutors than reading

the textbook, when the computer tutors provide video-based tutorials: "*I don't want to read books because when I read a book, I cannot concentrate [...] I spend more time reading a book than watching a video to get the concepts I want to learn.*" (P15, 5 years of programming experience).

Provide sufficient help — ICTs provide in-context resources within the system to help if needed. There are three specific functions that provide sufficient help for learners: (1) hint systems, (2) staff help, and (3) discussion panels. A hint system provide general help (usually generated automatically), and usually the very first type of assistance learners receive. P14 gave us an example on a hint system: "*you can just get the hints and they'll give you a hint and then you can either get the answer or you can just continue trying, but you're not just stuck there if you really can't figure it out.*" (P14, 4 years of programming experience).

Although the next two kinds of help require some type of human intervention, we report them since they were emergent themes. Currently, it appears that ICTs do not have the capability to supply some types of help that learners want (but humans can provide). However, this may change with advancements in natural language processing and machine learning, where systems might better detect and understand the context of their users' need for help [107, 190].

If the hint system fails to address learners' problems, some ICTs provide (human) staff help (also known as course experts). P1 gave us an example: "*They have two or three hints that they give, after that, they even say if you have any issues, we have [real] people who would help you out, and you can send your queries to them.*" (P1, 6 years of programming experience). Besides human staff help, discussion panels are built-in forums that provide a place for learners to discuss questions and ask for help. P9 described how 'community' in Dataquest helped her: "*Because in Dataquest, they also have something called 'the community', you can search [for] your questions*

in the community and the community members will post the answers. You can refer to their answers." (P9, 3 years of programming experience).

Designed for various learner levels — Some ICTs provide different pathways based on skill levels. It is helpful for learners to find courses matching their experience. P16 said: "*For Codecademy, they have levels, like 'did you just start learning code,' 'you already have some experience,' 'you're an expert.' So that helps because if you already know some coding you don't need a simple example because it's too easy.*" (P16, 4 years of programming experience).

Interestingly, one participant mentioned that he liked the feature of Codecademy which locks access to next module until he finished the current module. He had one year of programming experience, and he emphasized many times his anxiety as a beginner. This feature forced him to learn step-by-step. He told us: "*I really like this design, because this way can force me to do all the tasks in order one by one from easiest to the difficult one. I think this is really good, and I like it.*" (P5, 1 year of programming experience). Another participant mentioned that he liked the short video tutorials provided by Treehouse. The short videos relieve the cognitive load of learners when compared with a long video tutorial.

4.4.2 RQ1b: What do learners dislike about learning programming from ICTs?

Content design — This again refers to the course materials and the way ICTs organize and deliver information. Although 15 out of 20 participants liked the content provided by ICTs, there were also participants who did not like the content design. Those who did not like it thought that the tutorials and practice exercises were too basic to be useful. They liked ICTs but wished they could provide more advanced content. P2 considered himself as having a good understanding of programming basics

since he had 4 years of programming experience. He explained his concern to us: "*I was doing C++ [online], but I kind of stopped because I thought it was too easy and too basic. They just teach you such basic concepts and they don't go in-depth.*"

Another reason for the dislike was that the sections were redundant. One participant mentioned this to us: "*At the beginning, I find they are pretty useful, but like 4-5 lessons after, I find the content to be very dry, meaning it's really the same thing over-and-over again, I'm not really learning a lot of things that weren't [covered] there before.*" (P13, 2.5 years of programming experience).

We noticed that those who considered the content to be too basic were experienced learners (who had at least 3 years of programming experience), however, other junior-level learners mentioned that sometimes, information was too brief to understand sufficiently. Some ICTs only provide short introductions without really explaining the logic behind the material. P11 started programming 2 years ago, and was struggling to understand the complex logic behind certain concepts. "*For me, I don't like reading introductions [online], because they want to simplify their content and the introduction is so brief. Sometimes I don't fully understand the [programming] language,*" she said.

Locks access to more advanced concepts — Some ICTs require learners to finish the current module to unlock the next one. While we mentioned that one participant liked this feature earlier, four participants disliked this feature. All of them had prior programming experience and had clear goals on what they had to learn. Their learning efficiency was limited by this feature. P13 was learning programming for fun during his leisure time at work. He had been learning programming for 2 years. He complained to us: "*I already know what this is, and I want to skip it to [go to] the next module. I'm not able to do that, because I got to complete the first module, and then go to the second module. So, I didn't find that to be very user-friendly.*" If

that situation happens, the learner would try gaming the system [16], P19 shared her experience to us: "*Sometimes I just randomly select and if it's wrong, I'll just move on to the next options and then I don't really pay attention on the multiple choices questions.*" (P19, 3 years of programming experience).

Does not provide sufficient help — Although 9 out of 20 participants reported they could get sufficient help from ICTs, other participants held a different opinion. These participants reported that ICTs could not guide them to understand the logic behind problems effectively. P5, a novice programmer who often got stuck on problems in Codecademy, shared his experience with us: "*They [Codecademy] will actually show me the right answer. I still don't know what's wrong with my answer and it didn't show me or highlight the mistakes that I made, so I still don't know the answer.*" (P5, 1 year of programming experience).

4.4.3 RQ2a: What do learners like about learning programming from teachers?

Has real life programming experience – Teachers who are willing to share their real-life programming experience were favored by our participants. These experiences include: how to avoid common mistakes, how to style code, tips on interviews, and how to become a good programmer. P5 was a beginner in programming, he said: "*They [professors] always try to tell you how to avoid mistakes.*" Another example is P11, who had 2 years of programming experience, but was anxious about being a novice. She enjoyed learning from her teachers. "*They teach some things about the languages and they also tell some real experience for coding, and even some tips about interview and future working. They told us how a good programmer should do their job,*" she said. One participant observed that teachers are not only experienced in programming, but also in teaching. "*I think they not only have the real experience*

on using them [programming languages], but also have some experience on teaching, because they have a lot of students learn in this class, if they have problems, they will ask. So, the professor will know which part that most people think is difficult, so they will pay more attention on that part," He said. (P17, 6 years of programming experience).

Provides solid learning experience — Participants believed that they could learn programming with teachers more concretely and systematically than from ICTs. Teachers introduce new concepts along with background information and logic about these concepts. Teachers also help students stay on track. As an experienced programmer with 10 years' programming experience, P4 suggested beginners to start programming with a good teacher. He said: "*I believe a good teacher will teach you knowledge in a systematic way. If you have zero knowledge, the best way is to learn from a teacher, because if you learn from an online app, your knowledge is scattered, and it's not systematic. You learn piece-by-piece, [so] you might miss some bigger parts.*"

Provides conversations — Learners pointed out that conversations with teachers are invaluable because they can discuss ideas, explain their specific problems, and have someone to relate and/or look up to. P17 was an experienced programmer, and was full of project ideas that he liked to discuss with his advisor. He said: "*When you communicate with him, firstly you can solve your problem. And secondly if you have some ideas, you can talk with him and since he's experienced, he'll give you some feedback on your ideas and you know how to improve yourself or your program.*" (P17, 6 years of programming experience).

Another three participants thought that conversations let teachers better understand students' problems. Since learners can use different methods to express themselves in-person (e.g., drawing, writing, gesticulating), face-to-face conversation

with experts is a more efficient way to get problems solved than exchanging emails or searching for answers elsewhere. P10 told us that she could show her code directly to teachers when communicating face-to-face, so that the teachers can better understand her questions and help her line-by-line (P10, 2 years of programming experience).

Provides real-time help — When in class with teachers, learners can usually have their questions answered immediately. It is common to have bugs and errors when learners program. However, if these errors or bug are not solved immediately, it may lead to other issues that cause the learner to get stuck. ICTs often cannot provide real-time, customized help for specific questions, while teachers can. P11, a beginner in programming pointed out this advantage to us:"*I think it is better with a teacher. Because if there are any questions you can immediately ask for help.*"(P11, 2 years of programming experience). Another more advanced programmer, P16, said: "*[Learning] in-person is more instant, and I can do some things right away and get out the way whatever question I have.*" (P16, 4 years of programming experience).

4.4.4 RQ2b: What do learners dislike about learning programming from teachers?

Not efficient — Most participants reported that learning from teachers was less efficient. Teachers usually take more time in assigning practice and giving feedback, while online tools do these immediately and on-demand. As P2 told us: "*Because at the same time a teacher, you don't get assignments as quickly as you would online. So, the feedback comes in once a week as opposed to maybe you could literally do the whole course in a day if you want.*" (P2, 4 years of programming experience).

The teacher's lecturing style can also be less efficient than reading the course materials by learners themselves. P6 had 15 years of programming experience. Most of his teachers would just read straight from the textbooks. He expressed his

frustration with these types of teachers since he could just read from the textbooks himself at home: "*Most of the teacher just repeat the content of the book. They just read out the paragraph, the statements in a book. I think we can learn this from a book much faster than learning from a class. So, you can read the book and then practice by yourself.*"

In addition, three participants felt that it was easier to access materials through ICTs than teachers. For example, P13 stated that online materials could be accessed immediately, while for teachers, he needed to register and pay for a course first, and even be physically present in the class (P13, 2.5 years of programming experience).

Does not follow pace with students — Learners also have a big concern about teachers' speed of instruction. 9 out of 20 participants reported their experience of being unable to keep up with their teachers' pace. This occurred when the teachers delivered the content too quickly, or when students had difficulties understanding some content, but the teachers kept moving forward. For example, P20 was a novice programmer with only one year of experience; she once had a fast-paced programming course and could not keep up with the teacher's progress, so she turned to online courses to learn the same content. She said: "*And with class, things go by so quickly, we met only once a week and we have to cover so much. So, I feel like I'm lagging behind, I'm not catching up fast enough with the professor in the class, so I went to do something online where it can go at my own pace.*"

Two participants had the opposite experience—they found classes were far behind their progress. P16 was a student who always learned things quickly, and so, she often felt bored when she took a class but was ahead of the teacher's pace. She said: "*The class gets boring, because you already know. [...] there are students around you that are still asking questions and they don't get it, it's very hard for them to understand.*" (P16, 4 years of programming experience)

Provides inflexible curriculum — The content taught in a class were not always what learners expected. P12 had learned programming for years. He recalled his experience in college, when he selected a C++ course, expecting to learn something advanced, but the teacher only taught basic concepts. He told us how disappointed he was: "*what he taught during lecture, I already know, and what he taught was just the basic syntax, but he did not introduce those advanced [content] which [...] I already learned from another way. That's why I say he is not very helpful.*" (P12, 11 years of programming experience). While P10, who was less experienced than P12, told us that she expected to learn something basic, but what she got in class was too advanced to understand. She said: "*I figured I will get a tutor to teach me the basics, because he [professor] didn't teach us the basics.*" (P10, 2 years of programming experience).

Not responsive — Some participants complained that they lacked teacher attention in large classes. "*They (professors) are always busy; your problems might not be solved in time,*" P18 said (P18, 6 years of programming experience). Teacher's personal style may also be attributed to the lack of responsiveness. P17 told us one of his teachers who never replied his emails, he said: "*one of my professors never replied [to] my e-mails. The only way you'll find him is in his class. So, I will only have limited chances to ask questions.*" (P17, 6 years of programming experience).

Has insufficient teaching competencies — Some of the participants questioned the teaching competencies of some programming teachers, and there are three main types that refer to teaching competency. 3 out of 20 participants thought that the teachers they had were not experienced at programming. Just as P9 said: "*Maybe he is very experienced, but he is not experienced in programming, all of his code is just copy paste from another website, and then share it to you. He can't explain any details to you.*" (P9, 3 years of programming experience)

Participants also had concerns that their teachers' skills were not up-to-date since programming skills and technologies are constantly changing. For example, P12 had a solid foundation in programming; he found that teachers in school did not satisfy his needs because his goal was to always learn about the newest technologies. "*I think the teacher is usually far behind the current progress [...] But what I want to learn, is always something new,*" he said (P12, 11 years of programming experience).

Additionally, one participant shared her experience when she had a programming teacher who brought her personal emotions into teaching and made her upset: "*I always felt that particular teacher had some personal [issue] against me. I always felt that she deliberately did not grade me as much as I deserved. [...] she just asked me to leave her room, which is very rude. I don't know why. [...] It did matter to me for some time. I was kind of hurt and I was sad.*" (P1, 6 years of programming experience).

4.4.5 RQ3: What do learners think are the pros and cons of feedback from ICTs and teachers?

Pros of feedback from ICTs — The biggest positive feature of ICTs is the immediacy of feedback. Getting feedback immediately helps learning-by-doing. P5 just started learning programming and he believed that getting feedback immediately was important. He said: "*I think the most useful feature is that you can test immediately and get feedback. That's the most important one.*" (P5, 1 year of programming experience). P6 was more experienced than P5 but expressed something similar, saying: "*When you use an interactive tool to learn the programming language, you can get feedback immediately [...] You can learn it immediately, so that you can improve your programming skill very fast.*" (P6, 15 years of programming experience).

Some ICTs can provide step-by-step feedback, which is good in a way that the machine lets learners reflect on the errors as much as possible before giving them

the correct solutions. Just as P19 told us: "*The application lets you rethink about one point by giving you only a little bit of information to help. And if you still don't know the answer, they will ask you whether you want a hint.*" (P19, 3 years of programming experience). An even better interactive system can highlight the learners' errors, which helps them pinpoint errors quickly. P17 elaborated this point: "*They will show the result for your wrong code, and the result for the right code. And they will let you know which part you did wrong. So, it's quite clear. I think this feedback is useful. [...] If I did something wrong, it will highlight that specific part.*" (P17, 6 years of programming experience).

Cons of feedback from ICTs — The major problem of ICTs is that they often only tell the learners whether the final output is right or wrong. When learners generate the wrong output, the systems only tell them that they are wrong but do not specify where in the code an error exists (or when they do, it is typically a list of compile-time or run-time errors with an unhelpful error message), and why it is wrong. For example, P15 told us: "*If I did the exercises correctly, it will let you know you are correct. If I did something wrong, it will tell me to try again, [with] no specific instructions.*" (P15, 5 years of programming experience).

Even when learners generate the right output, ICTs will often only tell them that they are right, but will not give any additional feedback that a human teacher might (e.g., suggestions about coding style or alternative ways to solve a problem). P8 was a software engineer and knew the importance of a program's running speed in real-world scenarios and how important coding style can affect a group's efficiency. He said: "*There's code that's faster and then there's also code that's more efficient and then there's also code that takes up less space and you're basically looking for the [most] efficient one where it takes into account time and space. I mean the Treehouse*

and all these coding websites don't give you that kind of feedback." (P8, 5 years of programming experience).

Pros of feedback from teachers — Compared with ICTs, participants expressed that teachers' feedback were more specific and focused. As mentioned above, participants thought that it was important not only to understand where their code errors were, but also why it was an error so that they can better learn how to resolve it and avoid similar errors in the future. P5 told us a general impression on teacher's feedback: "*The professor is more specific and is more accurate. You can ask your professor to check your code line-by-line, and then he will point out your error, and will explain more. Maybe he will give you a reference.*". (P5, 1 year of programming experience).

P7 had a nice and patient teacher who gave him detailed explanations for his problems. He said: "*For SQL, I think at that time most of the problems we had is when we had to join table, and so she'd try to help me to understand. She will basically just draw out a table and then show me what my wrong code would produce as the output, and how it would be different from my input.*" (P7, 6 years of programming experience).

In addition, teachers can suggest and discuss different solutions of a problem with the students. Students can learn better solutions that can make their code run more efficiently. P3 gave us a comparison between feedback from ICTs and from teachers: "*You write a sorting algorithm and you think it is correct. If you tried on the website, it is also right, you think it is a perfect solution, but for the professor, he will tell you that there's a better sorting algorithm, so you will learn something better. But for the online tutorial, it only tells you if it is right or not.*" (P3, 6 years of programming experience).

Cons of feedback from teachers — Five participants mentioned that it took a long time to get feedback from their programming teachers. If learners do not get feedback in a timely manner, the benefits of learning-by-doing might not be realized since students might have forgotten about the specifics of the task(s) by the time they get the feedback. P8 shared his thoughts with us: "*With the teacher setting, you're submitting an assignment, waiting a week for them to look at it. So that it's a long process. Like it takes a while just to even know what you did wrong.*" (P8, 5 years of programming experience).

4.4.6 RQ4: In terms of learning programming, do learners prefer to learn from ICTs or teachers?

Prefer to learn from ICTs — 11 out of 20 participants reported that they preferred learning from ICTs. Among the 11 participants, one had 1-year programming experience, and the others had 3 or more years of experience. Most of them were self-identified as having good knowledge in programming basics. During the interview, most of them showed confidence on their self-learning abilities. Participants expressed that they liked to study at their own pace and in a more efficient way. In this sense, ICTs are better than teachers. This reasoning was more commonly seen with our experienced programmers. For example, P4 had 10 years of programming experience; when he learned a new skill or function, he aimed at applying it quickly to solve real problems. For him, learning from a teacher from the basics was not as efficient and flexible as learning from computer tutors. He said: "*Because I don't have time to spend a whole hour to sit in a classroom to take a course. Learning from a teacher, the time is not flexible, usually, I would prefer that I can learn this language this morning and I can use it this afternoon. It is not efficient to learn from a teacher. A teacher will teach you language from scratch. I already know some common sense about programming language, and the teacher will not personalize the course for you.*"

P20 was a more junior programmer than P4, but she had some foundation in programming. The interactive tool she used gave her a positive experience in learning programming, she expressed her preference for using it to learn: "*[I prefer] computer applications. It's sectioned so that it's easier to consume. And I can go at my own pace.with a class in the classroom, whether or not you get it, or you don't understand it, or you need some more time. The professor has to go. So, it's not at your own pace you can't like control the pace at which the class proceeds*" (P20, 1 year of programming experience).

Two participants thought that the skills and knowledge provided by ICTs were more up-to-date, while some teachers' skills might not. For example, P12 told us: "*I would prefer to learn through the [computer] application but not teacher. I think the teacher is usually far behind the current progress, because a teacher needs to learn this programming language first and then he can teach you. But what I would learn, is basically always something new.*"(P12, 11 years of programming experience). P19, a more junior programmer held the same view. She was a working professional; her work required her knowing new skills. She felt that what programming skills she learned at school could not apply to the problems she encountered at work, whereas what she learned from ICTs were always helpful. She said: "*The stuff you learn from school was more like standard ones, that everybody needs to learn. That's the basics. But actually, whatever you're doing in work or in life, it's totally different than what you learned from school. It doesn't really help.*" (P19, 3 years of programming experience).

Prefer to learn from teachers — Three participants expressed that they preferred to learn programming from teachers. They all had 1-2 years of programming experience. During the interview, they tended to be more anxious about any questions dealing with learning programming. They thought that they needed more expert help

to guide them through the process of learning programming. For example, P11 told us: "*I personally prefer someone to tell me. Not just that I go to read. For coding, I prefer following some examples, it doesn't matter if it's with the teacher or with the video, but I think it is better with a teacher. Because if there are any questions you can immediately ask for help.*" (P11, 2 years of programming experience).

Prefer to learn from both in combination — Instead of learning programming from a single method, some participants said that they preferred to learn from both in combination. They liked to learn from computers because they could grasp basic concepts efficiently and at their own pace. But they also liked to learn from teachers because they could discuss questions and advanced ideas with them. They agreed that a combination would be ideal. P1 elaborated her preference: "*[I] like a hybrid kind of thing, take a course online and then just meet my professor once a week to discuss what issues I had or just shoot out an e-mail saying that 'I was doing this particular section and I feel this could be done this way.' But just sitting there and just talking with the machine, I think it feels less personal. So, if it's a hybrid thing, you have the feasibility of doing the course and taking the course whenever you want to, and plus having the chance of speaking to a teacher gives you a broader platform to discuss your issues.*" (P1, 6 years of programming experience).

Preference depends on different situations — One participant said that his preference depended on his knowledge of the language. He said: "*It depends on what I try to achieve. If it's a new type of programming that I have no knowledge about at all, then I'll probably prefer to learn from a teacher. If I sort of know what it is about, I'll probably start with a computer-based method, and I'll go from there.*" (P13, 2.5 years of programming experience).

Another participant said that his preference for learning between an ICT or teacher depended on how deeply he wanted to learn. If he wanted to learn something

thoroughly, he preferred learning from a teacher; if he only needed to understand others' code at work, he would rather use an online interactive tool to understand the basics. He said: "*If I just want to know the programming language, so that I can understand other people's code, I think a computer application is good enough, because it teaches you basic syntax, basic logic, and I think that's all you need to be able to read and understand other people's code. But if you actually want to program by yourself, I would prefer taking a class with a professor.*" (P7, 6 years of programming experience).

One participant said that it depended on the teachers' teaching competency. If the teacher was experienced and willing to help, he would rather learn from them instead of an ICT. He said: "*If your professor is good at the language he is teaching, and if his skill is up to date, I think it's more helpful than learning yourself [with an ICT].*" (P18, 6 years of programming experience).

4.5 Discussion

We identified several features that learners like or dislike when learning programming from ICTs and teachers. Learners cared most about the following three factors: efficiency, feedback, and practice. We discovered that most of our participants' primary learning goal was to apply new programming skills quickly into their work or studies, so learning efficiency was their biggest concern. Most of them also believed that learning-by-doing was the best way to master programming skills, so immediate practice was an important factor to consider when choosing learning methods. In addition, due to the complexity of programming skills, learners preferred to get detailed feedback as quickly as possible. Designers of ICTs and teachers can benefit from this knowledge by focusing efforts on improving on these three aspects when teaching programming.For ICTs, the biggest strength, which most participants mentioned, was the embedded code editors and immediate practice, while a major

weakness was the content design (too basic, repetitive, or brief). To address this issue, designers could take advantage of code editors to provide more advanced, practice-oriented tutorials.

We also identified another factor to consider during our interviews, which was the existence of both basic and advanced levels of learners. While a few ICTs might separate their content for different experience levels of learners, most ICTs we are aware of do not; one beginner liked when the (ICT) system forced him to learn step-by-step (by locking content until finishing the current activity), whereas four, more experienced learners, did not like this feature. A key design consideration is to gauge a learner's experience at the beginning of the course/tutorial/activity so that the teacher or ICT can deliver content in a manner consistent with one's experience.

According to our findings, ICTs are efficient at delivering content with immediate practice, while teachers did a better job in providing customized help with real life experience. Both ICTs and teachers can benefit from these observations. First, ICTs can incorporate human experts to provide help when requested by online learners. Experts can also interact with learners in the system's online learning community, such as having conversations with learners or posting guides to address learners' concerns.

Second, we learned that teachers who are experienced in teaching were especially good at paying extra attention to students' needs when introducing content that students typically find difficult. ICTs can improve from this observation by gathering data from learners' learning logs in every course section (e.g., how many tries does someone take to write the correct code, or how much time do they spend on a concept) and provide extra instruction, help, or practice for the parts that most learners have difficulties with.

Third, our participants valued having conversations with teachers. They used these opportunities to gain coding tips, learn about real life experience as

programmers, exchange project ideas, and get help with questions. Listening to long lectures without minimal interaction was boring and meaningless to learners. Since we found that existing ICTs are good at delivering basic concepts and exercises, we believe it is beneficial to integrate them into programming classes to compliment teachers. Teachers can have ICTs deliver basic information (e.g., concepts, syntax) outside of class, and spend the time saved having more conversations with students regarding problems, projects, and real-life programming experience (e.g., using the "flipped classroom" model [175]).

Fourth, the comparison of feedback between ICTs and teachers suggests design opportunity for ICTs. It may not be feasible for teachers to give immediate feedback to students for all assignments/tasks, but ICTs can benefit from what we learned about feedback from teachers. This includes commenting on, giving suggestions, and showing alternative coding styles and run-time issues (such as code execution efficiency).

Finally, we identified two patterns from our participants, which were not anticipated from research questions but raised interesting questions to explore for future work. One is that experienced learners (who showed more confidence in learning) have higher learner autonomy [87] than beginners (who were more anxious about learning), meaning that experienced learners tend to use ICTs to conduct self-directed learning more often than beginners. Another pattern is that those with higher level of autonomy would prefer learning in an autonomy supportive system, such as a system that gives them more freedom to learn (see Subsection 5.4.2 "Locks Access to More Advanced Concepts"). While those with lower level of autonomy would prefer a system that can force them to learn (see Subsection 5.4.1 "Designed for Various Learner Levels").

These two emergent patterns indicate that beginners and experienced learners have different system preference for ICTs, and these differences are associated with

the level learner autonomy. This finding suggests a novel view for ICT designs that address the learner autonomy feature, which we will discuss in the following chapters.

4.6 Limitations

Our study has limitations that present further research opportunities. First, the snowball sampling method we used may have introduced a sampling bias. However, our participants represented a broad range of demographics, including years of experience with coding. Second, having 20 participants may raise questions about the representativeness of our sample and generalizability of our results. We reached data saturation [119]on our 16th interview and verified that our additional participants did not provide significantly different information from previous participants. Third, we found that there are factors that may affect how learners evaluate their learning experience, for example, learning environment (e.g., summer camp, college course, vocational training) and learning goals (e.g., learning for work, school practice, or personal inter-est). We will further investigate whether learning environments and learning goals, or even other factors (e.g., gender, age, job, level of experience, order of learning from a specific type of tutor), will cause effect on how learners evaluate their learning experience(s). Fourth, our study defined and examined ICTs within a specific subset of MOOCs. Furthermore, our study specifically examined only human teachers. There may be different types of ICTs and MOOCs that people have used to learn programming. Similarly, people may have learned from non-professional teachers or professors, such as informal tutors, friends, or relatives. These different types of ICTs/MOOCs and human teachers were deliberately excluded from this study to maintain a viable scope. However, our future work can include discussion about people's use of these other learning resources, which may reveal different findings and preferences from what was found here. Lastly, when presenting the quotes, we described the participants using their self-reported number of years

in programming (e.g., "P11 had 2 years of programming experience"). However, self-reported years of experience may not be a good indicator of participants' real programming ability or expertise. We will further study the relationship between years of experience and programming expertise. Other objective measures (e.g., test of knowledge) can be used to gauge learners' programming ability and experience level.

4.7 Summary

In this chapter, we explored learners' perspectives on receiving instruction from human teachers versus interactive computer tutors when learning programming. We found that efficiency and practice are the two main factors that learners care about when choosing between these two types of instruction, and the findings suggest the strengths and weaknesses of learning from interactive computer tutors and teachers, which we use as a basis for design suggestions for these types of instruction. We also find that learners with different autonomy level have different system preference, which raises future discussion about the design features to improve experiences for both kinds of learners.

CHAPTER 5

EXPLORING CS LEARNERS' AUTONOMY AND THEIR PREFERENCE FOR LEARNING SYSTEMS

5.1 Introduction

Innovation in technology continues to change at a rapid rate. Kruchten [100] coined the "five-year hypothesis," which posits that software engineers' key ideas become obsolete every five years. Given this situation, computer science (CS) learners have to continuously learn new skills to keep up with the ever-changing technology trends. Therefore, the knowledge they gain from formal education (e.g., from higher education institutions, which are often slow in changing curricula) may become obsolete or outdated fairly quickly. This requires those pursuing CS-related careers to be flexible and persistent—for them to be lifelong learners.

According to the literature, learner autonomy plays a vital role in developing lifelong learners [34]. Heloc [87] defines learner autonomy as the ability for one to control his/her own learning. The "control of learning" can be broken down into different aspects, including: setting goals, planning and executing learning activities, and evaluating the process [49, 114]. Contrasting traditional, teacher-centered pedagogy, autonomous learning requires classes to be student-centered, where teachers act in supportive roles to facilitate students' decision-making and problem-solving processes [104].

In past decades, learner autonomy has been frequently studied and promoted in the context of foreign language learning [172]. In Najeeb's [134] view, studying a foreign language is a lifelong endeavor, which neither starts nor ends in a language class, but something that must be constantly worked upon. This is very similar to the situation of CS learners. If they choose careers that deal with CS, they will have to continually work to maintain and update their knowledge. However, compared

with the long history of studying and researching learner autonomy in the context of foreign language, there are only a handful of similar works of language learning in computing education.

In order to gain a better understanding of their autonomy, our past work explored the experience of CS learners' thoughts about learning CS both online and in the classroom. We found that learners who showed a high level of autonomy felt that they were not supported by the educational system(s) or the educators. For example, they complained that the online systems they used did not give them enough freedom to explore on their own, and that their teachers often failed to help them achieve their learning goals. We also found that learners who showed a low level of autonomy felt that they needed extra guidance from their teachers. Our interview results also indicated some patterns, for example, that learners with more experience showed higher levels of autonomy than those with less experience, and learners with a higher level of autonomy preferred to study using an autonomy-supportive system while those with lower level of autonomy preferred to study using a non-autonomy supportive system.

Considering these past results and the role learner autonomy plays in developing lifelong learning, we believe that it is important to address the needs of CS learners with different levels of learner autonomy. Therefore, in this work, we describe results from an online questionnaire study designed to extend our past work and further explore CS learners' autonomy.Since we are using self-reported scale to gauge learner autonomy, the level of learner autonomy in this research also refers to the perceived level of learner autonomy.

Specifically, we ask the following research questions within the context of computing education:

1. What is the overall autonomy level of CS learners?

2. What factors contribute to the differences in CS learners' autonomy?

3. What kind of system designs are preferred by CS learners?

Understanding these questions can help us reflect on whether current pedagogies of computer science and educational systems can support learners in developing and maintaining autonomy, and how can we develop strategies to support the needs of CS learners with different levels of autonomy.

To explore these questions, we developed a survey and gathered 364 responses from CS learners. The results showed that our participants have overall high levels of learner autonomy, and that learners with more CS experience tend to have higher autonomy than learners with less CS experience. Our results also suggest that CS learners tend to prefer using autonomy supportive systems to study. By evaluating these results, we discuss implications for both computer science educators and educational system designers.

5.2 Background

5.2.1 Learner autonomy in general

Learner autonomy has been extensively discussed and applied in the context of education. Autonomy is broadly accepted as the "ability to take charge of one's own learning" [87], and many researchers have further elaborated autonomous learning in terms of a learner's ability to lead and control his or her own learning process and content. For example, Little [113] described autonomy as "a capacity for detachment, critical reflection, decision-making, and independent action". Dickinson [55] held that autonomy occurs when a learner becomes totally responsible for all of the decisions concerned with his or her learning and the implementation of those decisions. In modern education, where learners are more acclimated to reacting to established instructions, they develop autonomy when they independently set objectives of a learning program based on his or her status, which goes beyond established instructions or stimuli given by a teacher or instructor [27, 129, 128].

Personal initiative is key in autonomous learning when learners proactively engage with finding resources and opportunities, and become persistent and resourceful in learning [53, 146, 147]. Such ability to acquire knowledge or skills of value reflects the processes that learners' determine [39], or the psychological characteristic of individuals who are able to independently direct their learning [99, 124, 146]. To be more specific on what skills an autonomous learner should possess, Dam [49] detailed that a qualified autonomous learner independently chooses aims and purpose. Once these goals are set, he or she chooses supportive materials, methods, and tasks. The learner further exercises the choice and purpose made in implementing the selected plans. Upon accomplishment, an autonomous learner determines criteria for evaluation and complete self-assessment. In addition, Benson and Voller [21] discussed autonomy from its elements and concluded that it involves at least five phases, "for situations in which learners study entirely on their own; for a set of skills which can be learned and applied in self-directed learning; for an inborn capacity which is suppressed by institutional education; for the exercise of learners' responsibility for their own learning; for the right of learners to determine the direction of their own learning". No matter which forms of autonomy learners take in their education, autonomy is recognized as "a recognition of the rights of learners within educational systems" [22] and "the freedom to learn and the opportunity to become a person" [95]. The concept of learner autonomy is instructive in many fields of education. Not only is it a method to educate students, but also an objective of education; to give someone the ability to seek out, find, and internalize information to satisfy their curiosity—to become a lifelong learner.

5.2.2 Learner autonomy in natural language learning

The relationship between learner autonomy and natural language learning has been widely investigated as many researchers identified that autonomy is useful in learning

a new language and the ability to control learning experiences plays an important role in language education. Najeeb [134] suggested that independent language learning optimizes learner choice of decision making and targets on the needs of individual learners. With autonomy in language learning, learners develop skills in the target language individually, and benefit from the learning environments which are not directly mediated by a teacher nor in the interests of an institution. Learners' language proficiency is shown to be significantly and positively correlated with their autonomous level in language education [47, 127]. In examination of adult Japanese EFL learners in terms of autonomy, Mineishi [127] used learners' perception of autonomy to differentiate successful and less successful learners and found that successful learners are those who possessed autonomy proactively, while less successful ones are in the stage of achieving either active or reactive autonomy. Maftoon et al. [118] showed that good language learners tend to be more attentive to the class standards rather than cohering with the group, and all of the good language learners are autonomous. Razeq & Ahmad [1] suggested that training students to be autonomous learners contributes to and improves their English skills. With the understanding and implementation of autonomous learning strategies in and outside the classroom, students show improvement in English achievement and motivation to be responsible for their own learning. Furthermore, previous works have shown the importance of cultivating autonomy for English literature learning as it stimulates learners' interests and improves their strategies [69]. Abdipoor & Gholami [2] held that autonomous English learners tend to apply more language learning strategies than non-autonomous learners.

While there are obvious differences, learning a natural language and computer language also have some similarities (e.g., learning language rules, grammar, syntax, and semantics). In fact, several states in the USA have started allowing high school computer science courses to fulfill foreign language requirements for students [68].

With the similarities and the research of autonomy in language learning, we believe that the study of autonomy can be applicable in the context of computer science, and can benefit the community of computing education.

5.2.3 Learner autonomy in online learning

Distance (or online) learning has become increasingly popular in recent years. Firat [62] investigated the validity of autonomy concepts on distance learning environment and developed a scale to evaluate the e-learning autonomy of distance students who are responsible for their own learning. Based on the results, student autonomy in e-learning environments positively correlated to the level of ICT (Information, Communication and Technologies) but was not affected by enrolled program or gender. Lynch & Dembo [116] experimented on the five selected self-regulatory attributes (motivation, internet self-efficacy, time management, study environment management, and learning assistance management), and concluded that successful e-learning depends highly on five self-regulatory attributes which contribute to learners' different psychological processes to comprise their autonomy in online environment. In addition, Arnold [13] mentioned that the opportunities for learner autonomy can be created in online environments by introducing the eleven factors that promote autonomy in the online learning environment. Of the eleven factors, six of them ("peer learning, peer review, dialogue, reflection on learning, self-evaluation, and negotiated learning activities") are consistent with autonomy factors that were mentioned in the traditional learning literature while the other five (flexible access, learning facilitation, self-selection, lack of face-to-face contact, and media choices) were not identified as promoting factors in the previously mentioned literature. Although developing autonomy is important for online, self-paced students, whether the online environment can support learners' autonomy is still an open-ended question.

5.2.4 Learner autonomy in STEM

Learning and teaching in the fields of science, technology, engineering, and math is commonly referred to as STEM education [73]. STEM education has been discussed extensively internationally in the past decade [94], and engaging learners in STEM education has been shown to be important to society [29, 187]. Kelley & Knowles [93] reported that students of STEM fields lose interest in these areas when the connections to "crosscutting concepts and real-world applications" are missing in the learning process. In order to understand STEM learners, some works seek to understand learners in several fields of STEM. For example, in the field of Engineering, Chen et al. [36, 37] found several interesting results about learners' year of study in college, gender, race and college major in this context. His study demonstrated an effect of college students' year of study on their autonomous level as well as a positive relationship between Asian students' weaker self-beliefs and their autonomous learner traits. In addition, he also argued that engineering students gains autonomous learner traits through their engineering education journey while the differences of learner autonomy traits between their subgroups remain unclear. Similarly, Scott [157] confirmed that students' learner autonomy increases by the time they are in college while detailed correlation between autonomous level and year of study is unclear for biology college students. Although the term learner autonomy has not been commonly used in computing education studies, other similar terms, such as self-directed learning, have been explored. Boyer et al. [28] discussed how constructivism can improve programming pedagogy and student self-direction. They implemented a survey instrument on 15 students, and found that the majority of them were reported to have the ability to take control of their own learning, while surprisingly, students with less learning experience had higher self-directed learning scores than students with more learning experience. However, the small sample size of their study limited generalization. Mccartney et al. [122] interviewed 17

programming students and found that self-directed students are motivated to learn new programming topics, and that the students believed the benefits of self-directed learning when they join the workforce in the future. After the students join the workforce, it is more likely that they would learn new topics through online resources. Whether these online learning tools can support their self-directed learning remains an open question.

5.2.5 Learner autonomy in computing education

Through our literature review, we found that there are still several open questions of learner autonomy in the context of computing education. As the aforementioned work in foreign language learning suggested the positive relationship between learner autonomy and learning outcome, and considering the similarities between learning foreign language and computer science, we believe that autonomous learning can have positive impact on lifelong learning accomplishment of computer science (CS) learners. We were curious about CS learners' autonomy, and how this may affect their learning. In addition, previous work in e-learning and other STEM disciplines found attributes to differentiate autonomous learners, and our past work using interviews found differing needs based on CS learners' autonomy levels. In this work, we examine potential differences of autonomous learners and how to better support them through online learning tools.

5.3 Methodology

To study the autonomy and system preference of CS learners, we distributed a survey (See Appendix B for the whole survey) online and at a public university in the Northeast USA. The survey consisted of 3 sections. Section 1 had two parts, the first part collected the basic demographic information such as age, gender, ethnicity, degree and occupation. The second part was about participants' experience in learning

computer science related knowledge and skills. Since our target population was CS learners, the experience questions also served as filter questions. In our study, we apply the definition of computer science from Comer et al. [42] as "the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application." However, in the survey, we deliberately left the definition of computer science open to our participants, as we determined in past studies that using a strict/specific definition can potentially confuse or discourage participants who might consequently miscategorize themselves or self-select out of participation even though they meet our eligibility criteria. To ensure that we had a good representation of our target population, we included a free-response question asking them to list some of their computer science knowledge and skills. Three researchers discussed the scope of computer science to determine which skills fit within the scope of computer science. For example, skills such as"Excel VBA" and"building website" were considered within the scope of computer science, while"Word and Powerpoin" and "browsing websites and social medi" were not. Section 2 was the learner autonomy scale. We adopted 8 items from the Learner Autonomy Scale developed by Macaskill and Taylor [117] and 4 items from the E-learning Autonomy Scale developed by Firat [62]. These items were measured using a 5-point Likert scale from 1-strongly disagree to 5-strongly agree. Section 3 included system design scenarios, which measured participants' preference for autonomy supportive or non-supportive system designs. These items were measured using a 5-point Likert scale from 1-very unhelpful to 5-very helpful. These scenarios were created through interview, observation, and literature review.

We collected 364 valid responses. These consisted of 189 online responses from Amazon Mechanical Turk using Google Forms, and 175 responses were from students at a US university using paper forms. Recent studies have suggested that Mechanical Turk yields high quality results when used with specific parameters [23], which we

utilized in our study. We chose to use both online and in-person data sources to get a broad perspective (e.g., a student-centric and non-student-centric point-of-view) about learning from both online and offline resources. After the data collection, we compared the demographic information from the two sources. Although the gender distribution was comparable between the two groups, we found that both age and ethnicity were significantly different between the online and in-person groups. The difference in age was expected, as the university campus where we surveyed is mainly comprised of college-aged students (typically 18-22 years old). However, we had not expected the difference in ethnicity, as our surveyed university is listed as one of the top in diversity in the country (we found that this was due to a higher-than-expected number of Asian students participating in the study on-campus, which we cannot explain as this was not typical of other surveys we have conducted on our campus).

5.4 Findings

Data from the two survey sources were combined and analyzed together using SPSS. Our 364 valid responses were composed of 35.7% females and 64.3% males, with ages ranging from 18 to 71 years old (mean=28.44, stdev=10). The ethnicity distribution is: White (42.6%), Asian (34.1%), African American (9.9%), Hispanic (8%), and other (5.4%).

For the Learner Autonomy Scale, we first conducted factor analysis using PCA with Varimax rotation, since we assume that the factors are not correlated. The factors converged in two iterations. We identified two factors that both have high internal consistencies. Referring to the Learner Autonomy scale developed by Macaskill & Taylor (2010), we named the two factors Independence of Learning, which reflects all core components of autonomous learning, and Time Management, which reflects issues of managing learning time. The factor loadings and scale reliability

score are shown in Table 5.1. Throughout the study, we refer to these two factors as subscales.

Table 5.1 Factor Loadings and Reliability Scores

Items	**Independence of learning**	**Time Management**
I determine my own learning goals	0.74	-
I am open to new ways of doing familiar things	0.73	-
I control my own learning process	0.76	-
I enjoy finding information about new topics on my own	0.78	-
Even when tasks are difficult, I try to stick with them	0.67	-
I take responsibility for my learning experiences	0.68	-
My time management is good	-	0.82
I plan my time for study effectively	-	0.80
Internal consistency		
Cronbach's Alpha (Total scale=0.848)	0.842	0.855

The scores for both subscales were calculated by averaging the individual score of the items loaded on them. The resulting mean scores are 3.98 for Independence of Learning (median = 4.00), and 3.50 for Time Management (median = 3.50). The result indicates that CS learners have an overall medium to high level of learner autonomy.

To understand what factors may affect learners' autonomy level, we conducted a series of tests. When comparing means, the learner autonomy scores of gender and ethnicity were not significantly different for both subscales. Using age and years of experience as independent variable to check the relationship with dependent variable, the autonomy level, we conducted linear regression and found that age and years of experience in learning computer science have significant ($p < 0.05$), positive, but very weak relationship with both subscales. The R^2 and p-values are shown in Table 5.2.

We asked participants to self-report their experience level in computer science on a four-level scale: beginner, intermediate, advanced, and professional. Since there are four groups, we used a one-way ANOVA test, with the result indicating that there was at least one group that has a significantly different autonomous mean score than

Table 5.2 The R^2 and p-values Between Age, Years of Experience, and Subscales

	Independence of learning		**Time Management**	
	R^2	**p-value**	R^2	**p-value**
Age	0.013	0.032	0.05	<0.001
Years of Experience	0.043	<0.001	0.014	0.024

other groups. Since the four level groups meet the assumption of homogeneity of variances, we used Turkey's HSD to conduct a Post-Hoc test. The results indicate that learners in the self-identified advanced and professional levels have significantly higher means than beginner and intermediate level for both subscales, which means that experience level has a positive relationship with autonomy level. The means for each group and the significant level are shown in Table 5.3.

Table 5.3 Significant Differences Between Experience Level for Each Subscale

Learner autonomy scales	**Group (mean)**	**Group in comparison (mean)**	**p-value**
Independence of Learning	Beginner (3.83)	Professional (4.28)	0.002
	Intermediate (3.97)		0.048
Time Management	Beginner (3.25)	Advanced (3.8)	0.002
	Intermediate (3.38)		0.018
	Beginner (3.25)	Professional (3.88)	0.004
	Intermediate (3.38)		0.026

To analyze whether autonomy level has relationship with system preference, we conducted linear regression for each subscale with each individual design. The results indicate that autonomy level has very weak relationship with system preference. Table 5.4 shows the R^2 and p-value for the regression analysis between autonomy level and system preference. To further analyze the system preference, we calculated the mean score for each individual design. We found that all of the five autonomy supportive system designs win higher score than the three non-supportive system designs. The mean scores for each system design scenarios are shown in Table 5.5. The results indicate that CS learners prefer to use an autonomy supportive educational system to learn.

Table 5.4 R^2 and p-values for the Regression Analysis Between Autonomy Level and System Preference

	Independence of learning		**Time Management**	
Supportive	R^2	**p-value**	R^2	**p-value**
For each new term in the course material, the system will provide you with links to explore the term.	0.226	<0.001	0.025	<0.001
You have a study log to manage your study progress, including your course list, material list and error logs. So that you can reflect on your learning whenever you want.	0.085	<0.001	0.014	0.026
The system allows you to set up your goals at first and recommends courses that can help you achieve your goals. You have the freedom to choose the course portfolio to accomplish your goals.	0.164	<0.001	0.008	0.090
At the end of each section, you will be given practice problems/tasks. All of the practice is optional, so you have the option to skip them if you prefer.	0.073	<0.001	0.021	0.006
In the curriculum, all of the sections are unlocked, so you can learn from any section in any order that you want.	0.114	<0.001	0.042	<0.001
Non-supportive				
At the end of each section, you will be given practice problems/tasks. All of these are mandatory, and you have to correctly finish each of them to proceed.	0.036	<0.001	0.018	0.100
In the curriculum, all of the sections are locked, you can only go on to the next section by finishing the current one.	0.011	<0.001	0.027	0.002
For each practice, the system only has one correct solution. You will have to produce the exact same answer to proceed.	0.014	<0.001	0.021	0.006

Table 5.5 Mean Scores for Each System Design Scenario

	Mean	**Std.**
Supportive		
For each new term in the course material, the system will provide you with links to explore the term.	4.18	0.80
You have a study log to manage your study progress, including your course list, material list and error logs. So that you can reflect on your learning whenever you want.	4.07	0.91
The system allows you to set up your goals at first and recommends courses that can help you achieve your goals. You have the freedom to choose the course portfolio to accomplish your goals.	4.04	0.97
At the end of each section, you will be given practice problems/tasks. All of the practice is optional, so you have the option to skip them if you prefer.	3.97	1.04
In the curriculum, all of the sections are unlocked, so you can learn from any section in any order that you want.	3.82	1.04
Non-supportive		
At the end of each section, you will be given practice problems/tasks. All of these are mandatory, and you have to correctly finish each of them to proceed.	3.59	1.04
In the curriculum, all of the sections are locked, you can only go on to the next section by finishing the current one.	3.06	1.22
For each practice, the system only has one correct solution. You will have to produce the exact same answer to proceed.	2.91	1.19

To compare system preference of learners with different autonomy level, we grouped them as learners with lower level of autonomy (autonomy score less than 3.0 with a total of 5.0), and learners with higher level of autonomy (autonomy score equal to or higher than 3.0 with a total of 5.0). We ran independent sample t-test for each system design scenario, the results indicate that learners with higher autonomy had significantly higher preference for autonomy-supportive system design, while in non-supportive designs, no significant results were found. The results are summarized in Table 5.6.

5.5 Discussion

To summarize our findings, we found that: 1) CS learners have an overall medium to high level of learner autonomy, 2) Experienced CS learners tend to have higher learner autonomy than beginners, 3) CS learners prefer using autonomy supportive systems when learning, and that 4) learners with higher autonomy would prefer autonomy-supportive system more than those with lower autonomy.

Our findings have implications for both computer science educators and educational system designers. Since CS learners have overall medium to high level of autonomy, educators could adjust their instructional techniques to better support learner's autonomy: 1) Respect students' goals. The very first core component of learner autonomy is that learners can set their own goals [87], so we suggest that in a classroom setting, the course goals should reflect the students' goals. Instead of setting the goals for the whole class, teachers can act in the role that assist the students in setting and achieving their own goals for the course. 2) Encourage students to reflect and access their own learning. Autonomous learners constantly evaluate their own learning. Teachers should provide opportunities for their students to participate in the evaluation process by encouraging them to discuss their problem-solving process frequently in class. 3) Allow freedom for students on course activities. Learning

Table 5.6 Comparison of System Preference Between Learners with Higher Autonomy and Lower Autonomy

	Lower Autonomy	**Higher Autonomy**	
Supportive	**means**	**means**	**p-value**
For each new term in the course material, the system will provide you with links to explore the term.	3.65	4.27	<0.001
You have a study log to manage your study progress, including your course list, material list and error logs. So that you can reflect on your learning whenever you want.	3.73	4.12	<0.001
The system allows you to set up your goals at first and recommends courses that can help you achieve your goals. You have the freedom to choose the course portfolio to accomplish your goals.	3.69	4.09	0.001
At the end of each section, you will be given practice problems/tasks. All of the practice is optional, so you have the option to skip them if you prefer.	3.55	4.04	<0.001
In the curriculum, all of the sections are unlocked, so you can learn from any section in any order that you want.	3.25	3.92	<0.001
Non-supportive			
At the end of each section, you will be given practice problems/tasks. All of these are mandatory, and you have to correctly finish each of them to proceed.	3.27	3.64	0.049
In the curriculum, all of the sections are locked, you can only go on to the next section by finishing the current one.	2.65	3.13	0.97
For each practice, the system only has one correct solution. You will have to produce the exact same answer to proceed.	2.55	2.96	0.472

activities are often associated with learning goals. If the autonomous learners have clear goals, they will be able to select the proper learning activities for themselves (with guidance from teachers). In the classroom, instead of assigning the same practice to the whole class, teachers can provide alternative practices for students to choose from, or they can allow students to create their own projects.

While CS learners have an overall medium to high level of autonomy, we also found that beginners are less autonomous than more experienced learners. According to Heloc [87], autonomous learning is not only learning a certain knowledge or skill, but also about learning how to learn. Therefore, beginners, in their first stage of learning computer science, can be trained or encouraged to be better autonomous learners. Teachers, instead of focusing only on course materials, can also teach the process of learning computer science: 1) Train students to set goals step-by-step. Autonomous learners know how to set goals. For beginners, they often have general goals such as "I will learn this programming language," but they might get confused about where to start. Once they are familiar with the learning process, they will better understand how to break down the goals into smaller ones such as "I will learn syntax first" and "I will learn logic next." 2) Train them to select resources. Once goals have been set, autonomous learners will collect and screen the resources to accomplish their goals. Beginners can be trained to select the resources for their learning goals, they should also be trained that when they have problems, where can find resources to solve them. 3) Train them for self-evaluation. To take full ownership of the learning process, learners should be able to self-evaluate their learning without much help from others. Beginners should be trained the evaluation criteria for computer science subjects and be encouraged by teachers to conduct self-evaluation.

Since we found that CS learners tend to prefer using autonomy supportive system to learn, we suggest some ideas for designing an autonomy supportive system for educational system designers: 1) Respect learners' goal(s). The goal of courses

should be specified in the course description. Learners who come with specific learning goals can input these goals (or a subset of goals) into a course selection search system, which could then recommend courses accordingly. 2) Be resourceful. According to Candy [34], autonomous learners are motivated and curious. An autonomy supportive system should satisfy learners' curiosity. We suggest that for each new term or learning objective in the course material, the system provide links that learners can use to further explore the concept(s). This feature had the highest preference in our system design scenario survey. 3) Allow maximum freedom. Some instructional systems require mandatory pathways for learners. For example, sections are locked, and learners can proceed only by completing sections one-by-one, even if the material is already familiar/known to the learner. For autonomous learners, systems should allow maximum freedom for them to choose sections, practice, and allow some flexibility for evaluation criterion.

For the results that learners with higher autonomy would prefer autonomy-supportive system more than those with lower autonomy, it has implications for system designers to deign online educational tools that address the needs of highly autonomous learners.

5.6 Limitations

Our work has limitations which we plan to address in future work. First, the criteria we used to determine participants' experience level was from a self-reported scale from beginner to professional. However, participants may have different criterion for these selections and therefore may have led to inconsistencies in our user groupings. For future work, we can use more objective measures such as quizzes to test the skill level of participants as an alternative measure to experience. We could use this measure to reexamine if it has relationship with autonomy level. Second, for the system design scenarios, we asked participants to rate system features on a usefulness scale based

on descriptions. Although these scenarios and descriptions came from real, existing resources that the participants may be familiar with, having participants actually try/interact with these features before rating them may yield different response and results. For future studies we can put the described designs into a usable system that participants can actually interact with before rating them. Finally, besides using the preference scale, we may be able to get a richer account of participants' views of specific features through qualitative methods such as think-aloud or interviews.

5.7 Summary

This chapter investigated learner autonomy in the context of computing education. We surveyed 364 computer science learners to gain insight into their autonomy levels and autonomy-supportive system preferences. Our results indicate that: 1) CS learners have an overall medium to high levels of learner autonomy, 2) Experienced CS learners tend to have higher learner autonomy than beginners, 3) CS learners prefer using autonomy-supportive systems to learn, and 4) learners with higher autonomy would prefer autonomy-supportive system more than those with lower autonomy. These results better inform CS educators how to adapt their teaching and teaching tools to better train and support autonomous learning, and learning system designers to design autonomy supportive system for learners.

CHAPTER 6

REDESIGNING AUTONOMY FEATURES OF ICTS THAT TARGET AT LEARNERS' DIFFERENT AUTONOMY LEVELS

6.1 Introduction

As we discussed in previous chapters, there are still many open questions about ICT based MOOCs. For example, who are the users and how do they perceive these systems? Some studies have investigated the positive relationship between interactive learning and learner's motivation [35, 98, 189]. However, there are limited studies that look at different types of learners and how they interact with these systems. Knowing these is important, because our goal is to redesign the features of ICT based MOOC to make it more general inclusive to the learners, which means that we need to consider the different needs of different types of learners, in order to improve their learning experience.

In our past studies, Study 1 (Chapter 3) explored the learners of ICT based MOOCs [160], and found that not only beginners use it for introductory programming courses; experienced learners also use it to refresh skills or learn a new language. In Study 2 (Chapter 4), in-depth interviews were conducted with 20 programming learners with various experience levels [163]. This study revealed a pattern where experienced learners showed more control over their learning process, and that they were selective on ICT based MOOCs that could allow them to control their own learning. On the contrary, less experienced learners showed more anxiety when learning, and required scaffolding instruction from an instructor, either a computer tutor or a human teacher. This pattern can be partly explained by learner autonomy theory [87]. Experienced learners showed higher level of autonomy than beginners, and therefore, prefer more control over their learning. This implies a design feature

that should be considered for ICT based MOOCs: they should account for learners with different autonomy levels.

To generalize this finding from the qualitative study, Study 3 (Chapter 5) was conducted, which was a quantitative study to explore CS learners' autonomy and their preference for autonomous feature on ICT based MOOCs [161]. In this study, we found that CS learners overall tend to have medium to high level of autonomy, and that learners with higher autonomy report preferring autonomy-supportive system more than those their lower autonomy counterparts.

Considering these past results and the role learner autonomy plays in developing lifelong learning, we believe that it is important to address the needs of CS learners with different levels of learner autonomy. In order to test how an autonomy-supportive feature might affect learners (based primarily their level of learning autonomy), we implemented a *level-jumping* feature into an online educational programming game (see Figure 6.1). This is in contrast to most online learning curriculums and MOOCs that we have encountered, which are often locked to a specific sequence where later parts of the course are inaccessible until earlier parts are completed. We tested the game with this new level jumping feature with 350 new users, tracking their progress through the game for one week (7 days) each, spanning a total of 1.5 months. The results indicate that learners with higher autonomy use this feature more than their lower autonomy counterparts.

6.2 Related Work

In the research for lifelong learning, the three terms: *Learner Autonomy (LA)*, *Self-directed Learning (SDL)* and *Self-regulated Learning (SRL)* are often been used interchangeably, as they share a lot in common. By definition, all of the three terms point out that learners take active part in their learning process [86, 87, 192]. However, these three terms differ in some dimensions.

Figure 6.1 A screenshot of the Gidget introductory programming game.

For example, autonomy can be seen as a design feature of the learning environment [102], in contrast, "the extent to which self-regulation resides in the person or in the activity of the person underlies considerable conceptual divergence in the literature on self-regulation" [121].

For LA and SDL, SDL can be viewed as a manifestation of learner autonomy in the sense that learners take responsibilities for all the decisions related to their learning [56]. While LA is described as an ability to control one's own learning, it is also seen either (or both) as a means or as an end in education [87]. SDL can be the best method to develop LA and that is demonstrated by some research in different context [3, 4, 58, 71]. LA informs a significant measure of independence from the control of the environment or learners' degree of freedom [71, 88], and it can be seen as a design feature of the learning environment [102]. Therefore, it is more suitable to be used in the context of this study.

However, it is not our goal in this research to discuss the difference among the three terms. Although they share commons and differ in dimensions, the research on supporting SDL and SRL also has meaning for the design of learning environment that encompass autonomous features. For this reason, in the following related work discussion, besides the research for LA, we will also discuss the work for supporting SDL and SRL.

6.2.1 Supporting autonomy in traditional pedagogy

As early as the emergence of the concept of learner autonomy, researchers have been discussing the models and ways to promote and support learner autonomy from teacher's perspective in a traditional pedagogy.

For example, Vygotsky [178] emphasized the social-interactive dimensions of the learning process. In this model, the teacher's role is to create and maintain a learning environment in which learners can be autonomous in order to become more autonomous. Thanasoulas [172] considered an autonomous learner to be someone who is aware of, and identifying, his/her strategies, needs, and goals as a learner, and having the opportunity to reconsider approaches and procedures for optimal learning. During this process, a teacher's role cannot be neglected. There has to be a teacher who will adapt resources, materials, and methods to the learners' needs and even abandon all this if necessary.

Mariani [120] proposed a challenge-support model for teachers to produce their own individual and unique teaching style. In this model, low challenge with low support would discourage autonomous learners in a way that is the same as high challenge with low support would do to non-autonomous learners. And that is the teacher's job to find the balance between the challenge and the support.

Besides the teacher's role, Najeeb [134] emphasized the importance of learning environment. She proposed that the learning environment should be comfortable

enough for learners to feel encouraged, so that they are more willing to experiment with various learning strategies and not be afraid to ask questions and assistance. Similarly, other studies also discussed that an autonomy-supportive environment can facilitate learners' preference for more difficult tasks, active engagement, and perceived competence [44, 153].

As for curriculum design, researchers also explored ways to promote autonomy. For example, Cotterall [45] proposed five principles to design an autonomy supportive curriculum, which include: 1)the course reflects learners' goals, 2) tasks are explicitly linked to a simplified model of learning process, 3) task reflects real-world scenarios, 4) discussion and practice should facilitate task performance, 5) the course promotes reflection on learning.

Empirical studies have explored the effects of autonomy-supportive teaching on learning outcomes and motivation. Furtak & Kunter [66] conducted a 2 x 2 factorial design experiment, and they found that learners in a low autonomy-supportive condition learned significantly more, perceived more choice, and rated instruction as more positive than did students in a high autonomy-supportive condition. However, this experiment has been carried out in a traditional classroom, the research question that effects of autonomy supportive environment on learning outcome and motivation has not been explored in an online learning environment.

Some studies explored the barriers of promoting autonomy between teacher and students. For example, a recent qualitative study conducted by Basri [19] found that mismatch between teacher and learner expectations, spoon-feeding tendencies of teachers, limited teacher autonomy and large classes are the main barriers. However, the barrier of promoting autonomy between ICT and learners has not been investigated.

6.2.2 Supporting autonomy in online environment

Self-Determination Theory (SDT) argues that humans have a natural need for control, or autonomy. SDT asserts that the need for autonomy is one of the three basic psychological needs (i.e., autonomy, competence, and relatedness) humans have for the sake of thriving in life [51, 153].

In the same era as Ryan & Deci [51, 153], Goodyear [75] discussed the ways to develop educational technology that support learner autonomy or learner-managed learning. He proposed that autonomous learners have strong influence on the educational technology design, since they can reject the technologies that do not meet their needs, therefore, educational technology should be designed toward learner-centered.

On the contrary, there are debates that argue that the freedom to choose between a lot of choices may overload, distract or even disorientate learners [38, 67, 77, 92, 156]. For novices and learners with less prior knowledge and lower self-regulatory skills, studies have found that it is hardly for them to choose materials and information that best fulfill their needs [38, 90, 156].

Therefore, balancing the control between the systems and the learners becomes a question for researchers and designers who are pursuing the design for autonomy supportive systems.

Fischer & Scharff [63] contrasted three types of computational approaches to self-directed learning, including: 1) Intelligent tutoring systems [184], which represent a teacher- or system-driven approach; 2) Interactive learning systems [142], which represents a learner-drive approach; 3) Domain-oriented design environments, which are intermediate between the other two approaches by providing a more distributed approach to interact with domain problems. In their work, they held the opinion that, to support self-directed learning, the learning environment should allow learners to

work on real problems and tasks of their own choice, and yet still provide them with learning support contextualized to their chosen problems [64].

Gorrissen et al. [76] examined learning outcome and learners' motivation in three different hypermedia environments, which include autonomous-supportive, learner-controlled and system-controlled. The results indicated that providing learners with autonomy-supportive learning system had some beneficial effects compared to system-controlled and learner-controlled learning environments.

Throughout the literature review, we can summarize that much work has been done to discuss how to promote autonomy supportive learning environment to facilitate lifelong learning. However, with the development of educational technology, it attracts and supports a broader learner population, among which, there is a group of learners who are considered non-autonomous learners, or learners with lower level of autonomy. Their needs for an educational technology has not been addressed adequately by researchers.

Based on the literature review and our prior studies, we want to know how learners with different autonomy level would act differently for a same feature. Therefore, the research question for this study is: How does the autonomy-supportive feature of skipping to any part of the curriculum affect learners (based primarily on their level of learning autonomy)?

6.3 Methodology

6.3.1 The Gidget educational game

We modified our free introductory coding game, Gidget *(www.helpgidget.org)*, for this study. Gidget has a total of 37 core levels in its curriculum, where each level teaches a new programming concept (e.g., variable assignment, conditionals, loops, functions, objects) using a Python-like, imperative language [106, 111]. The objective of each level is to fix existing code to help the game's protagonist pass 1–4 test cases (i.e., statements that evaluate to `true`) after running the code. Each level introduces at

least one new programming concept, becoming progressively difficult in subsequent levels. Therefore, users are exposed to more programming concepts the farther they progress through the game. Finally, the game also includes a set of help features to help players overcome obstacles while coding on their own [105, 111]. This includes a frustration detector that provides encouraging hints/messages to those that are struggling with a level [107], and also auto-generates additional levels covering the same concept(s) to provide additional practice [106].

Normally, the game follows a specific order of levels (i.e., curriculum), building on content from previous levels. While the user interface shows the sequence/map of all core levels in the game (and indicating the players' current level; see top of Figure 6.1), it only allows the player to jump back to any previously completed level (at any time during game play). Players can also jump forward to the last level they have reached sequentially, but no further. All levels are visualized on the map as circles, with completed levels shown as solid circles, incomplete levels shown as hollow circles, and incomplete exam levels (explained in [111]) shown as hollow circles with a check mark. Finally, the currently loaded level is indicated with the Gidget character (see Figure 6.2). Hovering the mouse cursor over an incomplete level does not show any visual change, and clicking on an incomplete level does not do anything.

For this study, we modified the level-selection interface to allow players to jump to any level in the game, regardless of level completion status. Placing the mouse cursor on any other level grays out the current level's Gidget character and places a solid Gidget character that slowly rocks back-and-forth on that level marker. Clicking on the rocking character immediately jumps the player to that level. In addition, to keep the overall experience consistent across all users of this study, we disabled the game's auto-generated extra levels (as described in [106]). This was to prevent cases where someone might jump to a difficult level, trigger the frustration detector, then offered multiple additional practice levels covering the same concept(s). Finally, we

specifically pointed out this level-jumping feature in the game's introductory on-boarding tutorial (which all players see the first time they load the game), explaining the user interface, interaction method, and the level-jumping feature.

Figure 6.2 Closeup of the level selection map.

6.3.2 Participant recruitment

Our goal was to observe if and how players would use the level-jumping features within the game. We evaluated our system with a group of 350 new users of the game. The sign-up screen asked users for their age, gender, e-mail address, a checkbox indicating whether they have prior programming experience, and a checkbox (with link to consent form) asking if they were willing to participate in a research experiment. We intentionally did not define "programming" or "programming experience" as we determined in past studies [161, 162] that using a specific definition could potentially confuse or discourage participants who might consequently miscategorize themselves or self-select out of participation even though they meet our eligibility criteria. Mirroring a previous study [161], we asked those who indicated that they had prior programming experience an additional question: how they would rate their programming experience level on a four-level scale (beginner, intermediate, advanced, and professional). We used this measure to assign each player a learner autonomy group, which includes 1 (low learner autonomy) for beginners and intermediate levels, and 2 (high learner autonomy) for advanced and professional levels based on our prior work [161], which showed that this measure was significantly correlated with learner autonomy. This prior study combined subsets of the Learner Autonomy Scale created by Macaskill and Taylor [117] and the E-learning Autonomy Scale developed

by Firat [62], and demonstrated that higher self-rating in programming experience has a positive relationship with autonomy level.

For this study, we only selected users that indicated they were 18+ years and willing to participate in a research experiment. Adapting the methodology from our prior studies [106, 107], we set the observation time to 7 days (168 hours) per user to have a consistent evaluation window for all users. To promote quick account creation, we did not collect other demographic information such as ethnicity, geographical location, or education level. Participants were required to read and digitally sign an online consent form that briefly described the study. We were intentionally vague in our description of the level-jumping feature, stating that we were "testing new navigational features" to minimize potential leading or biasing of participants to pay attention more to that specific part of the interface. However, we debriefed all participants of the study procedures 7 days after the end of their individual observation window, by e-mail.

6.4 Findings

We report on our quantitative results comparing our participants' outcomes—split by demographic and experience features—using nonparametric Wilcoxon rank sums tests, kruskal wallis test, or simple linear regression, with a confidence of $\alpha = 0.05$, as our our data were not normally distributed. For all post-hoc analyses regarding gender data, we use the Bonferroni correction for three comparisons: ($\alpha = .05/3 = 0.0167$).

The study included 350 participants (aged 18–58; median 20). As a whole, our participants were composed of 180 females (51.4%), 161 males (46%), and 9 "not listed" or "decline to state" (2.6%). In addition, 255 (72.9%) indicated their experience level as beginner or intermediate, and 95 (27.1%) indicated their experience level as advanced or professional. We operationalized our key dependent variables, *engagement* and *jumping*, as the number of levels completed and the number of times

the jumping feature was used, respectively. In this study, jumping is defined as a click on the level selection map (see Figure 6.2) that results in loading another level (regardless of its completion status). For example, a player on Level 6 clicking on Level 6 would not count as a jump (since another level is not loaded). However, a player on Level 1 clicking on Level 30, then on Level 1, would count as 2 jumps.

6.4.1 High learner autonomy players use the jumping feature more

We found that all learners used the level jumping feature at least 2 times, regardless of having low learner autonomy (range 2-17; median 3) or high learner autonomy (range 2-38; median 8). Looking at the data more closely, we found that there was a significant difference in the number of levels participants completed by autonomy level ($W = 34722, Z = 12.370, p < .05$), with the high autonomy learners using the feature more than their counterparts.

We believe that all learners jumped at least two times because this feature was specifically mentioned in the on-boarding game tutorial, and at the minimum, someone using the feature to jump forward (first jump), would need to jump back to their original level (second jump). Next, our finding that high autonomy learners use the jumping feature more often than their low autonomy counterparts verifies our hypothesis (based on our previous work in [161]) that those with more experience (and therefore higher learner autonomy) would use and benefit from this jumping feature. Unlike low autonomy (inexperienced) learners, who do not necessarily know much about the topic and therefore would be better served learning programming concepts in a sequenced curriculum, the goal of high autonomy (experienced) learners may be to review or improve on their existing programming skills, refresh their knowledge for concepts, and/or to look for programming resources. Therefore, they may be more likely to use the jumping feature to browse through the different parts of the curriculum quickly, being more in control of their learning.

6.4.2 Males use the jumping feature more

We found a significant difference in usage of the jumping feature by gender (χ^2(2, N=350)=17.226, $p < .05$). Doing post-hoc analysis with the Bonferroni correction, we found that males used the jumping feature significantly more overall than their female counterparts (W=42.307,Z=4.109, $p < .05/3$). This result was independent of low learner autonomy (χ^2(2,N=255)=6.1464, $p < .05$) or high learner autonomy (χ^2(2,N=95)=6.1583, $p < .05$) in programming.

This result was not too surprising, as prior research [31] has shown that compared to females, males are statistically more likely to use selective information styles (following the first promising information, then potentially backtracking) [126], have lower risk aversion (be less wary of consequences) [57], and more willing to tinker (playfully experiment) [20]. Based on this, we believe that our male players were more likely to use the jumping feature simply because it was available in the interface (and also mentioned in the tutorial).

6.4.3 Low autonomy (female) learners complete more levels

Next, we explored if there was a difference in the number of levels participants completed. This is not a completely fair comparison, as everyone may have encountered levels in a different sequence (with later levels being considerably more difficulty than earlier levels) because of the jumping feature.

We found that low autonomy learners completed significantly more levels compared to their high autonomy counterparts ($W = 15002.5, Z = -1.987, p < .05$). Further analysis revealed that there was a significant different in the number of levels completed by gender within the low autonomy group (χ^2(2, N=255)=43.3806, $p < .05$). A post-hoc analysis with the Bonferroni correction showed that the low autonomy group females completed significantly more levels compared to their male counterparts (W=-61.579,Z=-6.655,$p < .05/3$). We calculated a simple linear

regression to predict level completion based on jumping behavior. Within the low autonomy group, we found a positive relationship between these variables ($F(1, 253) = 255.290, p < .05$), $R^2 = .502$). Examining this more closely, we found that this effect was strongest with females, where females in the low autonomy group who jumped more often completed more levels ($F(1, 144) = 206.433, p < .05$), $R^2 = .589$).

This result supports our hypothesis discussed in Section 6.4.1. The goal of high autonomy (experienced) learners may be to review or improve on their existing programming skills, and/or to look for programming resources. If high autonomy learners were using the level jumping feature primarily to explore what programming concepts the game curriculum covered, it would explain why they did not necessarily stay to solve/complete those levels. On the other hand, a low autonomy (inexperienced) learner's aim in playing a programming game is more likely to learn new things, and most or all of the programming concepts would be new to them. Therefore, whether or not they jump through the curriculum, less experienced learners have more incentive to complete levels. Perhaps those low autonomy learners that jump around the levels have a better idea of what is coming next (and also gain additional insights from the broken, starting code each level provides), and therefore more successful in completing levels. Most surprisingly, although our female participants were most likely not to use the jumping feature, those that did went on to be the individuals that completed most (or all) of the game levels. Females who did decide to use the jumping feature may have jumped back and forth between levels as a comprehensive information processing problem-solving strategy [31, 126], where they used the jumping feature to preview what was coming up, thereby gathering fairly complete information about the entire system before proceeding.

6.5 Discussion

Our findings show that both high and low autonomy learners (particularly males), used the level-jumping feature, with the former using this feature significantly more than the latter. We also found that high autonomy learners tend not to complete the levels they jump to, and that they complete significantly fewer levels overall compared to their low autonomy counterparts. We also found that the few low autonomy female learners who used the jumping feature readily, also ended up completing more levels than any other group.

These results support our findings discussed in Chapter 3, Chapter 4, and Chapter 5, in which we found that learners with different autonomy levels have different needs when using ICTs to learn. Higher autonomy learners may have the goal to refresh skills efficiently, and therefore they tend to prefer the system to allow them jumping freely, while their counterparts have the goal to learn new things concretely, so they would complete more levels.

In this study, higher autonomy learners used jumping feature more, while lower autonomy learners completed more levels, which means that satisfying the need of the former does not necessarily dissatisfy the latter. Therefore, designers for online resources teaching programming may benefit from allowing all users to skip around and explore the curriculum, instead of locking them into a specific sequence.

They may also do well in encouraging more of their learners (especially females) to use these types of jumping features to have them preview and better prepare for what is coming later in the curriculum.

6.6 Limitations and Future Work

We have several limitations to our study. We recruited participants who opted into a research study while signing up for an educational game. These participants may

already have high motivation, and therefore may not be completely representative of the larger population.

Next, we asked participants to self-report their own programming expertise. Participants may have different criterion for these selections and therefore may have led to inconsistencies in our user groupings. The groupings themselves may not account for all the different nuances of experience and/or learner autonomy. For future work, we could use more objective measures such as quizzes to test the skill level of participants as an alternative measure to experience.

In addition, in Study 3 (see Chapter5), we found that CS learners tend to have medium to high level of learner autonomy, while in this study, the participants skewed towards low autonomy learners. The reason might be that the tool we used, Gidget, is a system that teaches free introductory coding concept, which attracts more beginners than experienced learners. However, attracting more beginners is the current status for most interactive learning systems (see Section 2.2). For future research, we will evaluate the use cases for both beginners and experienced learners within the same system, and choose a system that attracts both learners to conduct further research.

Finally, our study results show that both high autonomy and low autonomy learners use the level-jumping feature (presumably to preview levels), and that although low autonomy users are less likely to utilize this feature, those that do are especially successful in completing more levels (particularly females). Our future work will examine these outcomes in more detail, and gather complementary qualitative data, to isolate the factors that are causing these effects.

6.7 Summary

In this chapter, we examined how an autonomous feature (jumping) affects learners with different autonomy levels. We found that 1) learners with higher autonomy used

the autonomous feature more, and 2) learners with lower autonomy completed more levels. We then discussed the implication of these findings.

CHAPTER 7

DISSERTATION SUMMARY

Learning interactively is fundamental and important for knowledge acquisition and skill development [18], especially for skills like programming, which requires extensive practice and making mistakes [5]. With the large demand for technology workers all around the world, and the lack of programming educators, interactive computer tutor (ICT) based MOOCs become popular alternative or supplement for traditional classroom lectures [171].

However, there is handful research about this kind of system, especially from learner's perspective. This work started out from understanding the learners of ICTs, and extended to the system redesign that would improve learner's experience.

7.1 Summary of Key Findings

This research is multidisciplinary, which consists of human computer interaction (HCI) and computing education. The goals of this research are:1) understanding the learners of ICT based MOOCs, and 2) exploring how to improve learners' experience with these systems. To reach the goals, we have conducted four studies using mixed methods. Except for the first study, which was an exploratory study, each study was built upon the findings of previous studies.

The first study was a content analysis study (see Chapter 3). The goal was to understand the users (mostly the learners) of Codecademy (a well-known ICT based MOOC that delivers programming courses). We reviewed 218 "Codecademy" related answers on Quora (a well-known Q&A website), and analyzed the textual data using inductive analysis. In this study, results indicated that 1) beginners were the main users, while experienced learners also used it to learn new languages or refresh skills, 2) it was good at delivering skills that can be visualized, such as web development/front

end courses, 3) interactive environment increased the engagement of learners, and 4) learners complained that the courses are not practical. This study uncovered that both beginners and experienced learners used ICTs to learn, but they had different goals. To further explore the difference between beginners and experienced learners, we conducted Study 2.

The second study was an interview study (see Chapter 4), in which we interviewed 20 programming learners about their experience with and preference between ICTs and teachers. Learners' perceptions for both learning methods were found and discussed, including their likes and dislikes. Some patterns were also found, such as 1) experienced learners had higher autonomy than less experienced ones, and 2) learners with higher autonomy would prefer learning from autonomy supportive system, and vice versa. The implications for both ICT designers and CS educators were discussed, and the patterns we found inspired next study.

In the third study (see Chapter 5), we investigated CS learners' autonomy using questionnaire built upon the findings from Study 2. It was distributed to 364 CS learners, scaling their autonomy level and system preference. The results indicated that: 1) CS learners had an overall medium to high levels of learner autonomy, 2) Experienced CS learners tended to have higher learner autonomy than beginners, 3) CS learners preferred using autonomy-supportive systems to learn, and 4) learners with higher autonomy would prefer autonomy-supportive system more than those with lower autonomy.

For the fourth study, we implemented an autonomous feature within an ICT based debugging game. The effect of this feature on learners with different levels of autonomy was examined. We found that: 1) learners with higher autonomy used the autonomous feature more, and 2) learners with lower autonomy completed more levels.

7.2 Design Implications

Educational technology should be designed toward learner-centered because learners can reject the technologies that do not meet their needs [75]. Researchers have been exploring ways to design for learners, and most of their work focused on increasing engagement and improving learning outcome (see Chapter 2). This work provides another unique view to design a learner-centered ICT, which is designing for learners with different autonomy.

Based on what we have done, we know that learners had complaints for current systems that do not give them freedom to learn, and that autonomous feature can affect both high and low autonomy learners in a positive way, satisfying autonomous learners will not necessarily dissatisfy non-autonomous learners. In addition, since non-autonomous learners will become autonomous learners eventually, when they become experienced learners (see Chapter 5), we can infer that all learners will need a certain level of autonomy within an ICT some time in their learning process.

The design implication for ICTs is that the system can add more autonomous features to serve learners with different autonomy. Although, the degree to which the system should provide the autonomy worth discussion, we will discuss that in the section of future work (see Section 7.4).

7.3 Contribution

The contributions of this dissertation are:

1. Among the first to explore ICT based MOOCs from the learners' perspective;
2. Providing a unique view to classify programming learners, which is from the level of learn autonomy;
3. Identifying autonomous needs of different programming learners;
4. Applying different autonomous settings to improve learners' learning experience;
5. Contributing to the field of human-computer interaction, computing education and learning science that would be useful to users, researchers, and designers of interactive learning systems for programming education.

7.4 Future Work

While the research questions for this dissertation have been answered, the results from our studies opens up a wide possibility of extended work for the future.

First, we tested the jumping feature in Study 4, and got some objective data such as the number of clicks and the number of completed level. The results were consistent with our previous findings, which are that high autonomy learners used this feature more and low autonomy learners completed more levels. For future work, we can get the learners' ratings of perceived usefulness and perceived ease of use to find out their subjective views about this feature. And furthermore, in-depth interviews or focus group can be done to understand the reasons behind the frequent use.

Second, learning outcome is always an indicator for a good learning system. The reason that we did not use this indicator in this dissertation is because learning outcome is not necessarily associated with learner's satisfaction, since learners had different goals when using ICTs to learn (see Chapter 3, and Chapter 4). Although the focus of this dissertation is all about learners, including who they are, why do they use and what do they think, for future work, the learning outcome can be tested together with the perceived usefulness and perceived ease of use. The correlation between these indicators can be checked, and control experiment can be conducted to compare these three indicators between the system with autonomous features and those without.

Third, we tested only one autonomy-related feature in this dissertation. However, according to our results from Study 3 (see Chapter 5), there are additional autonomy-related features that can be tested to observe their effect on different types of users.

Finally, although we found that autonomous feature can satisfy autonomous learners without necessarily dissatisfying non-autonomous learners, in the literature, there are debates that argue that the freedom to choose between a lot of choices

may overload, distract or even disorientate learners [38, 67, 77, 92, 156]. For novices and learners with less prior knowledge and lower self-regulatory skills, studies have found that it is hardly for them to choose materials and information that best fulfill their needs [38, 90, 156]. For future studies, we can put more focus on the learners with less autonomy, and explore their preference(s) between autonomous features and non-autonomous features. The findings will give us ideas and justifications to design more generally inclusive interactive computer tutors to benefit a broader range of learners.

APPENDIX A

INTERVIEW GUIDE FOR STUDY 2

In this appendix, you will see the recruitment ad that was used for recruiting participants, and the interview questions used for the study.

A.1 Recruitment Ad

Hi, My name is Ruiqi Shen and I am a PhD student at Department of Informatics at NJIT. I am currently conducting a research about what are learners' views about interactive computer-based tutors vs. human tutors, in terms of learning programming skills. An interactive computer-based tutor means a website or an application that you can learn programming interactively, with no or a little involvement of human teachers. This interview will take about 30 mins, I will do the audio recording throughout the interview.

A.2 Behavioral Questions

1) What is your occupation?

2) (If student) What is your major?

3) How long have you been learning programming in general?

4) What programming skills do you have?

5) How did you learn these skills? (online courses or applications? Human tutors? Digital or paperback books?)

6) When you have problems during learning programming skills, what will you do?

7) If you want to learn a new programming skill, what is the first method that comes to your mind? (why is that?)

A.3 Research Related Questions

1) Have you ever used any Computer-based tutors to learn a course or a skill?

a. What skill/course?

b. (If multiple, ask what's their most frequently used)

2) Could you describe the applications in detail?

a. How do you learn through this app?

3) What level of that course or skill are you learning from that application?

4) When you did something wrong with the questions. What feedback will you get?

a. How do you think about these feedback?

5) Have you ever got stuck in one problem and can't move on?

a. What will you do in this situation?

6) How did you hear about that application at first time?

7) Why do you think you need this application?

8) How long have you been using that application?

9) Did you complete that course/skill?

a. (If yes,) By completing that course, did you learn the knowledge and skill that you expected?

b. (If no,) What reasons caused you to quit?

10) Does the application provide you with all the help that you need?

11) Did you have to go elsewhere to get help?

a. (If no,) why not?

12) In what ways (if any), do you think this application helped you to learn?

a. Which features, or functions did you think were helpful for you to learn?

13) Did you experience anything negative when using this application to learn?

a. (If yes,) could you tell me more about that?

b. How did you deal with it?

14) In general, do you like this app or not?

a. (if yes, why) b. (if no, why)

15) To learn the knowledge/skill, have you tried any other methods?

a. (If yes,) could you tell me that method in detail?

b. (if no, why not?)

16) Do you plan to learn any programming skills recently?

a. (if no, do you plan to learn any programming skills in the future?)

b. (if yes, could you tell me more about that skill? (Such as levels, what is it used for etc.))

17) To learn this course/skill, which method do you plan to use (through digital applications, face to face with a teacher, with teachers online, read materials yourself, combination of those methods (what combinations?), or other methods you can think about?)

18) Have you ever learned programming from a teacher or a tutor?

a. Could you tell me more about this experience?

19) Did your teacher provide you with any feedback on your programming course?

a. What do you think the difference between feedback from a computer application and your teacher

20) In terms of learning programming, do you prefer to learn from a teacher, or to learn from a computer application

a. (If teacher, why, why not computer?)

b. (If computer, why, why not teacher)

21) Did you experience anything negative when learning from a teacher?

a. (if yes,) could you tell me more about that?

22) Do you like to communicate with your programming teachers?

a. (if yes,) In what ways do you like to communicate with you teachers?

b. Why (in that way)?

c. (If no, why not?)

A.4 Ending Questions

Is there anything that we didn't cover that you would like to talk about?

Just for the record, would you please tell me your age and race?

APPENDIX B

SURVEY QUESTIONNAIRE USED IN STUDY 3

In this appendix, you will see the survey questionnaire that was used for Study 3.

B.1 Demographics

1) What is your gender?

Female

Male

Prefer not to answer

Other:

2) What is your age?

3)Please specify your ethnicity

White

Hispanic, Latino, or Spanish origin

Black or African American

Asian

American Indian or Alaska Native

Middle Eastern or North African

Native Hawaiian or Other Pacific Islander

Some other race, ethnicity, or origin

4) What is the highest degree or level of school you have completed?

No schooling completed

Nursery school to 8th grade

Some high school, no diploma

High school graduate, diploma or the equivalent (e.g. GED)

Some college credit, no degree

Trade/technical/vocational training

Associate degree

Bachelor's degree

Master's degree

Professional degree (e.g. M.D., J.D.)

Doctorate degree (e.g. PhD)

5) What is your occupation?

B.2 Expertise in Computer Science

6) How many cumulative years have you learned/used computer science related knowledge/skills?

7) Please specify your level of computer science related knowledge/skills.

No experience

Beginner

Intermediate

Advanced

Professional

8) What are some of the computer science related knowledge and skills you've learned?

B.3 Learner Autonomy Scale

In the following statements, please rate from 1 (Strongly disagree) to 5(Strongly agree).

9) When I learn computer science: I determine my own learning goals.

I am open to new ways of doing familiar things.

I control my own learning process.

I don't evaluate my own studies.

I enjoy finding information about new topics on my own.

Even when tasks are difficult I try to stick with them.

I tend to be motivated to study by assessment deadlines.

I take responsibility for my learning experiences.

My time management is good.

I plan my time for study effectively.

I am happy studying on my own.

I don't arrange learning environments for myself.

For quality purpose, please select neutral for this line.

B.4 System Scenario Scale

In the following scenarios, please rate from 1 (Completely unhelpful) to 5(Completely helpful).

In the curriculum, all of the sections are unlocked, so you can learn from any section in any order that you want.

At the end of each section, you will be given practice problems/tasks. All of the practice is optional, so you have the option to skip them if you prefer.

If you get stuck on a practice problem/task, you can ask the system to provide you with the solution immediately instead of providing you with stepbystep hints.

You have a study log to manage your study progress, including your course list, material list and error logs. So that you can reflect on your learning whenever you want.

For each practice, the system only has one correct solution. You will have to produce the exact same answer to proceed.

At the end of each section, you will be given practice problems/tasks. All of these are mandatory, and you have to correctly finish each of them to proceed.

The system allows you to set up your goals at first, and recommends courses that can help you achieve your goals. You have the freedom to choose the course portfolio to accomplish your goals.

For each new term in the course material, the system will provide your with links to explore the term.

In the curriculum, all of the sections are locked, you can only go on to the next section by finishing the current one.

For quality purpose, please select neutral for this line

APPENDIX C

QUESTIONNAIRE FOR STUDY 4

In this chapter, you will see a short questionnaire used for Study 4

C.1 Questions

1. What is your gender?
2. What is your age?
3. Do you have any coding experience? yes/no
4. Please rate your programming experience level. Select from the following scale:

   beginner/intermediate/advanced/professional

5.Are you willing to participate in a research experiment?

   If yes — go to the consent form page If no — go to the game directly

REFERENCES

[1] Anwar Ahmad Abdel Razeq. University efl learners' perceptions of their autonomous learning responsibilities and abilities. *Regional Language Centre Journal*, 45(3):321–336, 2014.

[2] Neda Abdipoor and Hamid Gholami. Autonomous and non-autonomous efl learners' strategies and practices. *International Journal of Foreign Language Teaching and Research*, 4(14):107–121, 2016.

[3] Muhammad Madi Bin Abdullah, Sebastian Francis Koren, Balakrishnan Muniapan, Balakrishnan Parasuraman, and Balan Rathakrishnan. Adult participation in self-directed learning programs. *International Education Studies*, 1(3):66–72, 2008.

[4] Badli Esham Ahmad and Faizah Abdul Majid. Self-directed learning and culture: A study on malay adult learners. *Procedia-Social and Behavioral Sciences*, 7:254–263, 2010.

[5] Carlos Alario-Hoyos, C Delgado Kloos, Iria Estévez-Ayres, Carmen Fernández-Panadero, Jorge Blasco, Sergio Pastrana, and J Villena-Román. Interactive activities: the key to learning programming with moocs. *European Stakeholder Summit on Experiences and Best Practices in and Around MOOCs, EMOOCS*, 319, 2016.

[6] Vincent Aleven, Elmar Stahl, Silke Schworm, Frank Fischer, and Raven Wallace. Help seeking and help design in interactive learning environments. *Review of Educational Research*, 73(3):277–320, 2003.

[7] Heather Ames, Claire Glenton, and Simon Lewin. Purposive sampling in a qualitative evidence synthesis: A worked example from a synthesis on parental perceptions of vaccination communication. *BMC Medical Research Methodology*, 19(1):1–9, 2019.

[8] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The Journal of the Learning sciences*, 4(2):167–207, 1995.

[9] John R Anderson, Robert Farrell, and Ron Sauers. Learning to program in lisp. *Cognitive Science*, 8(2):87–129, 1984.

[10] John R. Anderson and Edward Skwarecki. The automated tutoring of introductory computer programming. *Communications of the ACM*, 29(9):842–849, 1986.

[11] John Robert Anderson. *The architecture of cognition*, volume 5. Psychology Press, East Sussex, United Kingdom, 1996.

[12] David Armstrong, Ann Gosling, John Weinman, and Theresa Marteau. The place of inter-rater reliability in qualitative research: An empirical study. *Sociology*, 31(3):597–606, 1997.

[13] Lydia Arnold. Understanding and promoting autonomy in uk online higher education. *International Journal of Instructional Technology & Distance Learning*, 3(7):33–46, 2006.

[14] Ryan Baker, Jason Walonoski, Neil Heffernan, Ido Roll, Albert Corbett, and Kenneth Koedinger. Why students engage in "gaming the system" behavior in interactive learning environments. *Journal of Interactive Learning Research*, 19(2):185–224, 2008.

[15] Ryan Shaun Baker, Albert T Corbett, Kenneth R Koedinger, and Angela Z Wagner. Off-task behavior in the cognitive tutor classroom: when students "game the system". In *Special Interest Group on Computer-Human Interaction Conference*, pages 383–390, 2004.

[16] Ryan Shaun Baker, Ido Roll, Albert T Corbett, and Kenneth R Koedinger. Do performance goals lead students to game the system? In *International Artificial Intelligence in Education Society (AIED)*, pages 57–64, 2005.

[17] Lecia J Barker, Charlie McDowell, and Kimberly Kalahar. Exploring factors that influence computer science introductory course students to persist in the major. *The ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin*, 41(1):153–157, 2009.

[18] Philip Barker. Designing interactive learning. In *Design and production of multimedia and simulation-based learning material*, pages 1–30. Springer, New York City, NY, United States, 1994.

[19] Fatma Basri. Factors influencing learner autonomy and autonomy support in a faculty of education. *Teaching in Higher Education*, pages 1–16, 2020.

[20] Laura Beckwith, Cory Kissinger, Margaret Burnett, Susan Wiedenbeck, Joseph Lawrance, Alan Blackwell, and Curtis Cook. Tinkering and gender in end-user programmers' debugging. In *ACM Conference on Human Factors in Computing Systems (CHI)*, pages 231–240, 2006.

[21] Phil Benson and Peter Voller. *Autonomy and independence in language learning.* Routledge, England, United Kingdom, 2014.

[22] Philip Benson. Autonomy as a learners' and teachers' right. In *Learner Autonomy, Teacher Autonomy: Future Directions*, pages 111–117. Longman, London, England, 2000.

[23] Frank R Bentley, Nediyana Daskalova, and Brooke White. Comparing the reliability of amazon mechanical turk and survey monkey to traditional market research

surveys. In *Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1092–1099, 2017.

[24] Mohammed Beyyoudh, Mohammed Khalidi Idrissi, and Samir Bennani. Towards a new generation of intelligent tutoring systems. *International Journal of Emerging Technologies in Learning (iJET)*, 14(14):105–121, 2019.

[25] Soly Mathew Biju. Taking advantage of alice to teach programming concepts. *E-Learning and Digital Media*, 10(1):22–29, 2013.

[26] Benjamin S Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.

[27] David Boud. *Developing Student Autonomy in Learning*. Routledge, England, United Kingdom, 2012.

[28] Naomi R Boyer, Sara Langevin, and Alessio Gaspar. Self direction & constructivism in programming education. In *9th ACM Special-Interest Group for Information Technology Education (SIGITE)*, pages 89–94. ACM, 2008.

[29] Jonathan M Breiner, Shelly Sheats Harkness, Carla C Johnson, and Catherine M Koehler. What is stem? a discussion about conceptions of stem in education and partnerships. *School Science and Mathematics*, 112(1):3–11, 2012.

[30] Peter Brusilovsky, Sergey Sosnovsky, Michael V Yudelson, Danielle H Lee, Vladimir Zadorozhny, and Xin Zhou. Learning sql programming with interactive tools: From integration to personalization. *ACM Transactions on Computing Education (TOCE)*, 9(4):1–15, 2010.

[31] Margaret Burnett, Simone Stumpf, Jamie Macbeth, Stephann Makri, Anicia Peters, and William Jernigan. Gendermag: A method for evaluating software's gender inclusiveness. *Interacting with Computers*, 28(6):760–787, 2016.

[32] J Bussgang and J Bacon. When community becomes your competitive advantage. *Harvard Business Review*, 2020.

[33] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological Methods & Research*, 42(3):294–320, 2013.

[34] Philip C Candy. *Self-Direction for Lifelong Learning. A Comprehensive Guide to Theory and Practice.* Education Resources Information Center (ERIC), Washington, D.C, United States, 1991.

[35] Yi-Hsing Chang, Jhen-Hao Hwang, Rong-Jyue Fang, and You-Te Lu. A kinect- and game-based interactive learning system. *Eurasia Journal of Mathematics, Science and Technology Education*, 13(8):4897–4914, 2017.

[36] John C Chen, Susan M Lord, and Karen J McGaughey. Engineering students' development as lifelong learners. In *120th American Society for Engineering Education Annual Conference and Exposition Proceedings*, 2013.

[37] John C Chen, Karen McGaughey, Susan M Lord, et al. Measuring students' propensity for lifelong learning. In *23rd Annual Conference of the Australasian Association for Engineering Education*, page 617, 2012.

[38] Sherry Y Chen, Jing-Ping Fan, and Robert D Macredie. Navigation in hypermedia learning systems: experts vs. novices. *Computers in Human Behavior*, 22(2):251–266, 2006.

[39] Adele Chene. The concept of autonomy in adult education: A philosophical discussion. *Adult Education Quarterly*, 34(1):38–47, 1983.

[40] Ryan Chmiel and Michael C Loui. Debugging: from novice to expert. *The ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin*, 36(1):17–21, 2004.

[41] Ylona Chun Tie, Melanie Birks, and Karen Francis. Grounded theory research: A design framework for novice researchers. *SAGE open medicine*, 7:2050312118822927, 2019.

[42] Douglas E Comer, Peter J Denning, David Gries, and Michael C Mulder. *Report of the ACM Task Force on the Core of Computer Science*. ACM, New York City, NY, United States, 1988.

[43] Juliet Corbin and Anselm Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage publications, Thousand Oaks, CA, United States, 2014.

[44] Diana I Cordova and Mark R Lepper. Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology*, 88(4):715, 1996.

[45] Sara Cotterall. Promoting learner autonomy through the curriculum: Principles for designing language courses. *English Language Teaching Journal*, 54(2):109–117, 2000.

[46] Natalie J Coull and Ishbel MM Duncan. Emergent requirements for supporting introductory programming. *Innovation in Teaching and Learning in Information and Computer Sciences*, 10(1):78–85, 2011.

[47] Deng Dafei. An exploration of the relationship between learner autonomy and english proficiency. *Asian English Language Education Publishing Journal*, 24(4):24–34, 2007.

[48] Brian LF Daku and Keith Jeffrey. An interactive computer-based tutorial for matlab. In *30th Annual Frontiers in Education Conference*, volume 2, pages F2D–2. IEEE, 2000.

[49] Leni Dam. *From Theory to Classroom Practice*. Authentik, Dublin, Ireland, 1995.

[50] Refsnes Data. w3school. com the world's largest web developer site, 2017.

[51] Edward L Deci and Richard M Ryan. The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological Inquiry*, 11(4):227–268, 2000.

[52] Omer Deperlioglu and Utku Kose. The effectiveness and experiences of blended learning approaches to computer programming education. *Computer Applications in Engineering Education*, 21(2):328–342, 2013.

[53] Marcia Gail Derrick. *The Measurement of An Adult's Intention to Exhibit Persistence in Autonomous Learning*. The George Washington University, Washington, DC, United States, 2001.

[54] Sneha R Deshmukh and Vijay T Raisinghani. Abc: Application based collaborative approach to improve student engagement for a programming course. In *IEEE 10th International Conference on Technology for Education (T4E)*, pages 20–23, 2018.

[55] Leslie Dickinson. Autonomy, self-directed learning and individualization. *English Language Teaching Journal*, 103:14–15, 1978.

[56] Leslie Dickinson. *Self-Instruction in Language Learning*. Cambridge University Press, Cambridge, England, 1987.

[57] Thomas Dohmen, Armin Falk, David Huffman, Uwe Sunde, Jürgen Schupp, and Gert G Wagner. Individual risk attitudes: Measurement, determinants, and behavioral consequences. *Journal of the European Economic Association*, 9(3):522–550, 2011.

[58] Fengning Du. Student perspectives of self-directed language learning: Implications for teaching and research. *International Journal for the Scholarship of Teaching and Learning*, 7(2):n2, 2013.

[59] Reinders Duit and Jere Confrey. Reorganizing the curriculum and teaching to improve learning in science and mathematics. *Improving Teaching and Learning in Science and Mathematics*, pages 79–93, 1996.

[60] Robert G Farrell, John R Anderson, and Brian J Reiser. An interactive computer-based tutor for lisp. In *The Association for the Advancement of Artificial Intelligence (AAAI)*, pages 106–109, 1984.

[61] Roberto B Figueroa and Emely M Amoloza. Addressing programming anxiety among non-computer science distance learners: A upou case study. *International Journal for Educational Media and Technology*, 9(1):56–67, 2015.

[62] Mehmet Firat. Measuring the e-learning autonomy of distance education students. *Open Praxis*, 8(3):191–201, 2016.

[63] Gerhard Fischer and Kumiyo Nakakoji. Amplifying designers' creativity with domain-oriented design environments. In *Artificial Intelligence and Creativity*, pages 343–364. Springer, New York City, NY, United States, 1994.

[64] Gerhard Fischer and Eric Scharff. Learning technologies in support of self-directed learning. *Journal of Interactive Media in Education*, 1998(2), 1998.

[65] Barbara A Fox. Cognitive and interactional aspects of correction in tutoring. *Teaching Knowledge and Intelligent Tutoring*, 1, 1991.

[66] Erin Marie Furtak and Mareike Kunter. Effects of autonomy-supportive teaching on student learning and motivation. *The Journal of Experimental Education*, 80(3):284–316, 2012.

[67] James E Gall and Michael J Hannafin. A framework for the study of hypertext. *Instructional Science*, 22(3):207–232, 1994.

[68] Gaby Galvin. Some say computer coding is a foreign language. *US News*, October 2016.

[69] Xu Gang. A case study on the effectiveness of learner autonomy in british and american literature study. *Studies in Literature and Language*, 10(1):88, 2015.

[70] Kenneth J Gergen. *An Invitation to Social Construction*. Sage, Thousand Oaks, CA, United States, 1999.

[71] Luk Gharti. Self-directed learning for learner autonomy: Teachers' and students' perceptions. *Journal of NELTA Gandaki*, 1:62–73, 2019.

[72] Paul Gill, Kate Stewart, Elizabeth Treasure, and Barbara Chadwick. Methods of data collection in qualitative research: interviews and focus groups. *British Dental Journal*, 204(6):291–295, 2008.

[73] Heather B Gonzalez and Jeffrey J Kuenzi. Congressional research service science, technology, engineering, and mathematics (stem) education: A primer. *Retrieved May*, 8:2018, 2012.

[74] Leo A Goodman. Snowball sampling. *The Annals of Mathematical Statistics*, pages 148–170, 1961.

[75] Peter Goodyear. Environments for lifelong learning. In *Integrated and Holistic Perspectives on Learning, Instruction and Technology*, pages 1–18. Springer, New York City, NY, United States, 2000.

[76] Chantal JJ Gorissen, Liesbeth Kester, Saskia Brand-Gruwel, and Rob Martens. Autonomy supported, learner-controlled or system-controlled learning in hypermedia environments and the influence of academic self-regulation style. *Interactive Learning Environments*, 23(6):655–669, 2015.

[77] Judith TM Gulikers, Theo J Bastiaens, and Rob L Martens. The surplus value of an authentic learning environment. *Computers in Human Behavior*, 21(3):509–521, 2005.

[78] Philip J Guo. Online python tutor: embeddable web-based program visualization for cs education. In *44th ACM Technical Symposium on Computer Science Education*, pages 579–584, 2013.

[79] Philip J Guo. Codeopticon: Real-time, one-to-many human tutoring for computer programming. In *28th Annual ACM Symposium on User Interface Software & Technology*, pages 599–608, 2015.

[80] Mark Guzdial. Limitations of moocs for computing education-addressing our needs: Moocs and technology to advance learning and learning research (ubiquity symposium). *Ubiquity*, 2014(July):1–9, 2014.

[81] Kevin A Hallgren. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in Quantitative Methods for Psychology*, 8(1):23, 2012.

[82] Michael J Hannafin. Interaction strategies and emerging instructional technologies. *Canadian Journal of Educational Communication*, 167, 1989.

[83] John Hattie. *Visible Learning for Teachers: Maximizing Impact on Learning.* Routledge, England, United Kingdom, 2012.

[84] Neil T Heffernan and Kenneth R Koedinger. An intelligent tutoring system incorporating a model of an experienced human tutor. *International Conference on Intelligent Tutoring Systems*, pages 596–608, 2002.

[85] Jan Herrington, Thomas C Reeves, Ron Oliver, and Younghee Woo. Designing authentic activities in web-based courses. *Journal of Computing in Higher Education*, 16(1):3–29, 2004.

[86] Roger Hiemstra. Self-directed learning. *The Sourcebook for Self-Directed learning*, 920, 1994.

[87] Henri Holec. *Autonomy and Foreign Language Learning.* Education Resources Information Center (ERIC), Washington, D.C, United States, 1979.

[88] Jing Peter Huang and Phil Benson. Autonomy agency and identity in foreign and second language education. *Chinese Journal of Applied Linguistics*, 36(1):7, 2013.

[89] Lon Ingram and Michael Walfish. Treehouse: Javascript sandboxes to helpweb developers help themselves. In *The Advanced Computing Systems Association (USENIX) conference on Annual Technical Conference*, pages 13–13, 2012.

[90] Slava Kalyuga. The expertise reversal effect. In *Managing Cognitive Load in Adaptive Multimedia Learning*, pages 58–80. IGI Global, Hershey, PA, United States, 2009.

[91] Oscar Karnalim and Mewati Ayub. The use of python tutor on programming laboratory session: Student perspectives. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 2(4):327–336, 2017.

[92] Judy Kay. Learner control. *User Modeling and User-adapted Interaction*, 11(1-2):111–127, 2001.

[93] Todd R Kelley and J Geoff Knowles. A conceptual framework for integrated stem education. *International Journal of STEM Education*, 3(1):11, 2016.

[94] Teresa J Kennedy and Michael R.L Odell. Engaging students in stem education. *Science Education International*, 25(3):246–258, 2014.

[95] Brian Kenny. For more autonomy. *System*, 21(4):431–442, 1993.

[96] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *ACM Conference on Innovation and Technology in Computer Science Education*, pages 41–46, 2016.

[97] Ada S Kim and Amy J Ko. A pedagogical analysis of online coding tutorials. In *ACM Technical Symposium of Special Interest Group on Computer Science Education (SIGCSE)*, pages 321–326, 2017.

[98] Kyong-Jee Kim. Motivational challenges of adult learners in self-directed e-learning. *Journal of Interactive Learning Research*, 20(3):317–335, 2009.

[99] Malcolm S Knowles. *The Modern Practice of Adult Education: From Pedagogy to Andragogy*. Cambridge Book Co., Cambridge, England, 1988.

[100] Philippe Kruchten. The biological half-life of software engineering ideas. *IEEE software*, 25(5):10–11, 2008.

[101] Mikko-Jussi Laakso, Teemu Rajala, Erkki Kaila, and Tapio Salakoski. Novice learning. In Norbert M. Seel, editor, *Encyclopedia of the Sciences of Learning*, pages 2482–2483, New York City, NY, United States, 2012. Springer.

[102] Terry Lamb and Hayo Reinders. *Learner and Teacher Autonomy: Concepts, Realities, and Response*, volume 1. John Benjamins Publishing, Amsterdam, Netherlands, 2008.

[103] Kris MY Law, Victor CS Lee, and Yuen-Tak Yu. Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, 55(1):218–228, 2010.

[104] Lina Lee. Blogging: Promoting learner autonomy and intercultural competence through study abroad. *Language Learning & Technology*, 2011.

[105] Michael J Lee. How can a social debugging game effectively teach computer programming concepts? In *ACM The International Computing Education Research (ICER) Conference*, pages 181–182, 2013.

[106] Michael J Lee. Auto-generated game levels increase novice programmers' engagement. *The Journal of Computing Sciences in Colleges*, page 70, 2020.

[107] Michael J Lee. (re)engaging novice online learners in an educational programming game. *The Journal of Computing Sciences in Colleges*, 35(8), 2020.

[108] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. Principles of a debugging-first puzzle game for computing education. In *2014 IEEE Symposium on Visual Languages and Human-centric Computing (VL/HCC)*, pages 57–64. IEEE, 2014.

[109] Michael J Lee and Amy J Ko. Personifying programming tool feedback improves novice programmers' learning. In *7th International Workshop on Computing Education Research*, pages 109–116, 2011.

[110] Michael J Lee and Amy J Ko. Comparing the effectiveness of online learning approaches on cs1 learning outcomes. In *11th Annual International Conference on International Computing Education research*, pages 237–246, 2015.

[111] Michael J Lee, Amy J Ko, and Irwin Kwan. In-game assessments increase novice programmers' engagement and level completion speed. In *ACM The International Computing Education Research (ICER) Conference*, 2013.

[112] Sang Hwa Lee, Junyeong Choi, and Jong-il Park. Interactive e-learning system using pattern recognition and augmented reality. *IEEE Transactions on Consumer Electronics*, 55(2):883–890, 2009.

[113] David Little. Learner autonomy. *Dublin*, 86:11, 1991.

[114] David Little. Freedom to learn and compulsion to interact: promoting learner autonomy through the use of information systems and information technologies. *Taking Control: Autonomy in Language Learning*, 1:203–218, 1996.

[115] Tharindu Rekha Liyanagunawardena, Karsten O Lundqvist, Luke Micallef, and Shirley Ann Williams. Teaching programming to beginners in a massive open online course. *OER14: building communities of open practice*, 2014.

[116] Richard Lynch and Myron Dembo. The relationship between self-regulation and online learning in a blended learning context. *The International Review of Research in Open and Distributed Learning*, 5(2), 2004.

[117] Ann Macaskill and Elissa Taylor. The development of a brief measure of learner autonomy in university students. *Studies in Higher Education*, 35(3):351–359, 2010.

[118] Parviz Maftoon, Parisa Daftarifard, and Morvarid Lavasani. Good language learner: from autonomy perspective. *Linguistic and Literary Broad Research and Innovation*, 2(1):104–114, 2011.

[119] Kirsti Malterud, Volkert Dirk Siersma, and Ann Dorrit Guassora. Sample size in qualitative interview studies: guided by information power. *Qualitative Health Research*, 26(13):1753–1760, 2016.

[120] Luciano Mariani. Teacher support and teacher challenge in promoting learner autonomy. *Perspectives: A Journal of TESOL Italy*, 1997.

[121] Jack Martin and Ann-Marie McLellan. The educational psychology of self-regulation: A conceptual and critical analysis. *Studies in Philosophy and Education*, 27(6):433–448, 2008.

[122] Robert McCartney, Jonas Boustedt, Anna Eckerdal, Kate Sanders, Lynda Thomas, and Carol Zander. Why computing students learn on their own: Motivation for self-directed learning of computing. *ACM Transactions on Computing Education*, 16(1):2, 2016.

[123] Barbara Means, Yuki Toyama, Robert Murphy, Marianne Bakia, and Karla Jones. Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies. *Centre for Learning Technology*, 2009.

[124] Sharan B Merriam, Rosemary S Caffarella, and Lisa M Baumgartner. *Learning in Adulthood: A Comprehensive Guide*. John Wiley & Sons, Hoboken, NJ, United States, 2006.

[125] Douglas C Merrill, Brian J Reiser, Michael Ranney, and J Gregory Trafton. Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *The Journal of the Learning Sciences*, 2(3):277–305, 1992.

[126] Joan Meyers-Levy and Barbara Loken. Revisiting gender differences: What we know and what lies ahead. *Journal of Consumer Psychology*, 25(1):129–149, 2015.

[127] Midori Mineishi. East asian efl learners' autonomous learning, lerner perception on autonomy and portfolio development: In the case of educational contexts in japan. *International Journal of Arts and Sciences*, 3(17):234–241, 2010.

[128] Michael Grahame Moore. Learner autonomy: The second dimension of independent learning. *Convergence*, 5(2):76, 1972.

[129] Michael Grahame Moore. The theory of transactional distance. In *Handbook of Distance Education*, pages 84–103. Routledge, England, United Kingdom, 2013.

[130] Briana B Morrison and Betsy DiSalvo. Khan academy gamifies computer science. In *45th ACM Technical Symposium on Computer Science Education*, pages 39–44, 2014.

[131] Robert Moser. A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it. In *2nd Conference on Integrating Technology into Computer Science Education*, pages 114–116, 1997.

[132] Zahraa Muhisn, Mazida Ahmad, Mazni Omar, and Sinan Muhisn. The impact of socialization on collaborative learning method in e-learning management system (elms). *International Journal of Emerging Technologies in Learning*, 14(20):137–148, 2019.

[133] Robert Murphy, Larry Gallagher, Andrew E Krumm, Jessica Mislevy, and Amy Hafter. Research on the use of khan academy in schools: Research brief. *Stanford Research Institute (SRI) International*, 2014.

[134] Sabitha SR Najeeb. Learner autonomy in language learning. *Procedia-Social and Behavioral Sciences*, 70:1238–1242, 2013.

[135] Eric Nersesian, Margarita Vinnikov, Jessica Ross-Nersesian, Adam Spryszynski, and Michael J Lee. Middle school students learn binary counting using virtual reality. In *IEEE Integrated STEM Education Conference (ISEC)*, pages 1–8. IEEE, 2020.

[136] Scott Nicholson. A recipe for meaningful gamification. In *Gamification in Education and Business*, pages 1–20. Springer, New York City, NY, United States, 2015.

[137] Pratya Nuankaew. Dropout situation of business computer students, university of phayao. *International Journal of Emerging Technologies in Learning*, 14(19):115–131, 2019.

[138] Pamel Ochieng. An analysis of the strengths and limitation of qualitative and quantitative research paradigms. *Problems of Education in the 21st Century*, 13:13, 2009.

[139] Marie Olsson and Peter Mozelius. On design of online learning environments for programming education. In *Academic Conferences and Publishing International ECEL*, pages 12–24, 2016.

[140] Daniel FO Onah, Jane Sinclair, and Russell Boyatt. Dropout rates of massive open online courses: behavioural patterns. *EDULEARN14 Proceedings*, 1:5825–5834, 2014.

[141] Steven Ovadia. Quora. com: another place for users to ask questions. *Behavioral & Social Sciences Librarian*, 30(3):176–180, 2011.

[142] Seymour A Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books, New York City, NY, United States, 2020.

[143] Laura Pappano. The year of the mooc. *The New York Times*, 2(12):2012, 2012.

[144] Sharoda A Paul, Lichan Hong, and Ed H Chi. Who is authoritative? understanding reputation mechanisms in quora. *arXiv Preprint arXiv:1204.3724*, 2012.

[145] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. *Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 204–223, 2007.

[146] Michael Kamano Ponton. *The Measurement of An Adult's Intention to Exhibit Personal Initiative in Autonomous Learning*. The George Washington University, Washington, DC, United States, 1999.

[147] MK Ponton, PB Carr, and GJ Confessore. Learning conflation: A psychological perspective of personal initiative and resourcefulness. *Practice & Theory in Self-directed Learning*, pages 65–82, 2000.

[148] David Pritchard and Troy Vasiga. Cs circles: an in-browser python course for beginners. In *44th ACM Technical Symposium on Computer Science Education*, pages 591–596, 2013.

[149] Brian J Reiser, John R Anderson, and Robert G Farrell. Dynamic student modelling in an intelligent tutor for lisp programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 85, pages 8–14, 1985.

[150] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[151] Herbert J Rubin and Irene S Rubin. *Qualitative Interviewing: The Art of Hearing Data*. SAGE, Thousand Oaks, CA, United States, 2011.

[152] Martin Russell, Robert W Series, Julie L Wallace, Catherine Brown, and Adrian Skilling. The star system: an interactive pronunciation tutor for young children. *Computer Speech & Language*, 14(2):161–175, 2000.

[153] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67, 2000.

[154] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality & Quantity*, 52(4):1893–1907, 2018.

[155] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 207–214, 2005.

[156] Katharina Scheiter and Peter Gerjets. Learner control in hypermedia environments. *Educational Psychology Review*, 19(3):285–307, 2007.

[157] Graham W Scott, J Furnell, CM Murphy, and R Goulder. Teacher and student perceptions of the development of learner autonomy; a case study in the biological sciences. *Studies in Higher Education*, 40(6):945–956, 2015.

[158] Charles Severance. Khan academy and computer science. *Computer*, 48(1):14–15, 2015.

[159] Jason H Sharp. Using codecademy interactive lessons as an instructional supplement in a python programming course. *Information Systems Education Journal*, 17(3):20, 2019.

[160] Ruiqi Shen. Interactive computer tutors as a programming educator: Improving learners' experiences. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–2, 2020.

[161] Ruiqi Shen, Joseph Chiou, and Michael J Lee. Becoming lifelong learners: Cs learners' autonomy. *Journal of Computing Sciences in Colleges*, 35(8):267–267, 2020.

[162] Ruiqi Shen and Michael J Lee. Learners' perspectives on learning programming from interactive computer tutors in a mooc. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–5, 2020.

[163] Ruiqi Shen, Donghee Yvette Wohn, and Michael J Lee. Comparison of learning programming between interactive computer tutors and human teachers. In *ACM Conference on Global Computing Education*, pages 2–8, 2019.

[164] Zach Sims and C Bubinski. Codecademy. *http://www.codecademy.com*, 2011.

[165] StackOverflow. Stack overflow developer survey 2019, 2019.

[166] Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, and Christoph Meinel. Towards practical programming exercises and automated assessment in massive open online courses. In *IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 23–30. IEEE, 2015.

[167] Holly Stillson and John Alsup. Smart aleks... or not? teaching basic algebra using an online interactive learning system. *Mathematics and Computer Education*, 37(3):329, 2003.

[168] Anselm Strauss and Juliet M Corbin. *Grounded Theory in Practice*. SAGE, Thousand Oaks, CA, United States, 1997.

[169] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. Learning difficulties in programming courses: undergraduates' perspective and perception. In *International Conference on Computer Technology and Development*, volume 1, pages 42–46, 2009.

[170] Tan-Hsu Tan and Tsung-Yu Liu. The mobile-based interactive learning environment (mobile) and a case study for assisting elementary school english learning. In *IEEE International Conference on Advanced Learning Technologies*, pages 530–534, 2004.

[171] Terry Tang, Scott Rixner, and Joe Warren. An environment for learning interactive programming. In *45th ACM Technical Symposium on Computer Science Education*, pages 671–676, 2014.

[172] Dimitrios Thanasoulas. What is learner autonomy and how can it be fostered. *The Internet Teachers of English as a Second Language(TESL) Journal*, 6(11):37–48, 2000.

[173] David R Thomas. A general inductive approach for qualitative data analysis. *American Journal of Evaluation*, 2003.

[174] Kenneth Tobin, Deborah J Tippins, and Alejandro José Gallard. Research on instructional strategies for teaching science. *Handbook of Research on Science Teaching and Learning*, 45:93, 1994.

[175] Bill Tucker. The flipped classroom. *Education Next*, 12(1):82–83, 2012.

[176] Marisa Venter and Arthur James Swart. Continuance use intention of a gamified programming learning system. In *Annual Conference of the Southern African Computer Lecturers' Association*, pages 17–31, 2018.

[177] Jorge A Villalobos, Nadya A Calderon, and Camilo H Jiménez. Developing programming skills by using interactive learning objects. *ACM Special Interest Group on Computer Science Education (SIGCSE) Bulletin*, 41(3):151–155, 2009.

[178] Lev S Vygotsky. *Thought and Language*. MIT press, Cambridge, MA, United States, 2012.

[179] David Wade-Stein and Eileen Kintsch. Summary street: Interactive computer support for writing. *Cognition and Instruction*, 22(3):333–362, 2004.

[180] Jason A Walonoski and Neil T Heffernan. Detection and analysis of off-task gaming behavior in intelligent tutoring systems. In *International Conference on Intelligent Tutoring Systems*, pages 382–391, 2006.

[181] Gang Wang, Konark Gill, Manish Mohanlal, Haitao Zheng, and Ben Y Zhao. Wisdom in the social crowd: an analysis of quora. In *22nd International Conference on World Wide Web*, pages 1341–1352, 2013.

[182] Joe Warren, Scott Rixner, John Greiner, and Stephen Wong. Facilitating human interaction in an online programming course. In *45th ACM Technical Symposium on Computer Science Education*, pages 665–670, 2014.

[183] Robert S Weiss. *Learning from Strangers: The Art and Method of Qualitative Interview Studies*. Simon and Schuster, New York City, New York, United States, 1995.

[184] Etienne Wenger. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufmann, Burlington, MA, United States, 2014.

[185] Younghee Woo and Thomas C Reeves. Meaningful interaction in web-based learning: A social constructivist interpretation. *The Internet and Higher Education*, 10(1):15–25, 2007.

[186] Beverly Woolf and David D. McDonald. Building a computer tutor: Design issues. *Computer*, 17(09):61–73, 1984.

[187] Yu Xie, Michael Fang, and Kimberlee Shauman. Stem education. *Annual Review of Sociology*, 41:331–357, 2015.

[188] Aharon Yadin. Reducing the dropout rate in an introductory programming course. *ACM Inroads*, 2(4):71–76, 2011.

[189] Noriyasu Yamamoto. An interactive learning system using smartphone: Improving students' learning motivation and self-learning. In *9th International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 428–431. IEEE, 2014.

[190] An Yan, Michael J Lee, and Amy J Ko. Predicting abandonment in online coding tutorials. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 191–199, 2017.

[191] Ahmed Mohamed Fahmy Yousef, Mohamed Amine Chatti, Ulrik Schroeder, and Marold Wosnitza. What drives a successful mooc? an empirical examination of criteria to assure design quality of moocs. In *IEEE 14th International Conference on Advanced Learning Technologies*, pages 44–48, 2014.

[192] Barry J Zimmerman. A social cognitive view of self-regulated academic learning. *Journal of Educational Psychology*, 81(3):329, 1989.