Programming Learners' Perceptions of Interactive Computer Tutors and Human Teachers

https://doi.org/10.3991/ijet.v15i09.12445

Ruiqi Shen (✉), Donghee Yvette Wohn, Michael J. Lee
New Jersey Institute of Technology, New Jersey, USA
rs858@njit.edu

Abstract—People often learn programming in face-to-face courses or from online tutorials. Interactive computer tutors—systems that provide learning content interactively—are becoming more common in online tools such as those teaching computer programming. Studies have shown that teachers, interactive computer tutors, and the combination of both are efficient and effective in teaching programming. However, there is limited understanding of the comparative perspectives of those learning from these two different sources. We conducted an exploratory study using semi-structured interviews and recruited 20 participants with programming experience from both teachers and interactive computer tutors. Speaking with our participants, we surfaced factors that learners like and dislike about the two learning resources and discussed the strengths and weaknesses between the two. Based on our findings, we discuss implications for designs that programming educators and interactive computer tutor developers can use to improve their teaching effectiveness.

Keywords—Tutors, interactive computing tutors, student perspectives, computing education, human teachers

1 Introduction

Learning to program is considered a difficult process and requires continuous practice much like learning natural languages. However, unlike natural languages that can be used in different situations of everyday life, learners typically program within the constraints of a computer screen [1]. The difficult nature of programming may increase the dropout rate in both classrooms [2], [3] and massive open online courses (MOOC) [4]. In addition, the quality of programming teachers and MOOCs can also serve as a factor affecting dropout rates [5].

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 [6], but the lack of computer teachers continues to be a concern of researchers and educators [7]. 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 [8]. More recently, MOOCs have a 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 [9]. However, the limited amount of interaction with teachers and limited extrinsic motivators are major constraints and future development opportunities for MOOCs.

For the purposes of this paper, 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). Farrell et al. were among the first to introduce an interactive computer tutor (ICT) to teach programming [10]. They described it as a two-component system: a "problem solver" (which can interpret learners' code and provide feedback) and an "advisor" (which provides guidance to learners throughout the learning process). Systems for teaching programming such as Codecademy, Datacamp, and Treehouse are similar to the ICT described by Farrell, which include a problem solver and an advisor. We use Farrell's definition of ICTs in this paper and examine systems with these features. 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 [11], [12] and are gaining more popularity with learners [13], [14].

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 paper 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.

2 Related Work

2.1 Learning from interactive computer tutors

Mastering programming skills requires extensive practice and making mistakes [15]. Many MOOC websites, such as Codecademy and Khan Academy, integrate tutorials with extensive exercises using code editors with feedback systems [16]. Empirical studies show that students who learn programming interactively through well-designed computer systems can achieve good learning outcomes and higher self-efficacy [10], [12]. However, what are the good features of an ICT for delivering educational programming content? Early research by Reiser et al. 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 [17]. More recent work by Staubitz et al. 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) [18]. Pritchard & Vasiga summarized that built-in coding environments are beneficial for students' continuity in learning-by-doing [16]. While most educators will agree that a mentor is essential in the initial learning process for beginners, Liyanagunawardena et al. 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 [19]. Users can also identify the benefits of features such as deliberate instructional design (designed instructions), learning analysis (self-reflecting information), and instant feedback [20]. Our study expands on these works, aiming to explore whether these commercial systems (such as Codecademy and DataCamp)—which include features such as the ones mentioned above—can be considered good ICTs, and what users of these systems think about them.

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. Many studies have shown the effectiveness of human teachers [21], [22]. A teacher can guide students and can effectively time how much thinking a student should do before providing hints or answers [23]. This is especially important for novices, who can benefit more from interactions with teachers [24]. Teachers can also intervene at the right time to prevent students from becoming too frustrated [22], which is especially important in the early stage of learning, when learners have a higher likelihood of quitting [2]. Robins et al. 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 [25]. However, questions remain about what specific teaching methods contribute to an effective programming class. Pears et al.'s overview of programming classes found little systematic evidence to support any particular teaching approach that answers this question [26]. Tan et al. conducted a survey to gain insight on 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 [27]. However, questions such as what kind of lab sessions they like and whether they could get sufficient consultation opportunities remain largely unknown to us.

2.3 Comparing / combining interactive computer tutors and human Teachers

Means et al. conducted an extensive literature review comparing online learning and face-to-face learning [28]. 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. 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 [22].

For teaching programming specifically, Warren et al. compared the feedback from both computer tutors and teachers in a classroom setting [29]. 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 proposed an intelligent 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 [30]. Deperlioglu & Kose 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 [31]. Beyyoudh et al.'s work also indicated that a combination of intelligent tutoring system and human tutors increased learners' motivation [32].

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?

3 Method

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 [33], [34], especially when there is limited research in the literature to gain an in-depth perspective into subjects' views and experiences [23]. We used a snowball sampling method to recruit our participants [35], 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 (US). 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 two-component definition for ICTs (see introduction and [10]). 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?")
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. in their work describing how to measure intercoder reliability for semi-structured interview studies [36]. 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 [37][38], 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 Results

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).

4.1 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.

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."

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.

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."

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."

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."

Flexible: This allows learners to go at their own pace whenever and wherever they want using ICTs. 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."

Participants also highlighted flexibility in location, such as P20, who said: "Like, 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." 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.

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."

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 Codeacademy, they just teach you each command. It's a faster way to learn it."

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
3. Discussion panels

A hint system provides 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."

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 [39], [40].

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."

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."

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."

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. 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.

4.2 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."

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 introduction[s 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."

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."

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. Teachers know about and can focus on parts that most students find difficult to understand.

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."

Another 3 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.

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." 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."

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."

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.

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.

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."

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." 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."

Is 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. 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."

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.

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." 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."

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." 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."

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."

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."

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 in their code error 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." 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."

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."

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."

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."

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." 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."

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 more 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."

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."

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."

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 thor-

oughly, 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."

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]."

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 programmer 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 [41]).

Lastly, 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).

6 Limitations and Future Work

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 [42] 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 interest). 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.

7 Conclusion

In this paper, 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. Our findings also 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.

8 Acknowledgement

This material is based upon work supported by the National Science Foundation under grants DRL-1837489 and IIS-1657160. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

9 References

[1] R. Moser, "A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it," in ACM SIGCSE Bulletin, 1997, 29:3, 114–116. https://doi.org/10.1145/268809.268853

[2] A. Yadin, "Reducing the dropout rate in an introductory programming course," ACM inroads, 2011, 2:4, 71–76. https://doi.org/10.1145/2038876.2038894

[3] P. Nuankaew, "Dropout Situation of Business Computer Students, University of Phayao," Journal of Emerging Technologies in Learning, 2019, 14:19. https://doi.org/10.3991/ijet.v14i19.11177

[4] D. F. Onah, J. Sinclair, and R. Boyatt, "Dropout rates of massive open online courses: behavioural patterns," EDULEARN, 2014, 5825–5834.

[5] R. Shen, D. Y. Wohn, and M. J. Lee, "Comparison of Learning Programming Between Interactive Computer Tutors and Human Teachers," in ACM CompEd, 2019, 2–8. https://doi.org/10.1145/3300115.3309506

[6] P. J. Guo, "Codeopticon: Real-time, one-to-many human tutoring for computer programming," in ACM UIST, 2015, 599–608. https://doi.org/10.1145/2807442.2807469

[7] M. Guzdial, "Limitations of MOOCs for Computing Education-Addressing our needs: MOOCs and technology to advance learning and learning research," Ubiquity, 2014. https://doi.org/10.1145/2591683
[8] J. R. Anderson and E. Skwarecki, "The automated tutoring of introductory computer programming," Communications of the ACM, 1986, 29:9, 842–849. https://doi.org/10.1145/6592.6593
[9] T. Tang, S. Rixner, and J. Warren, "An environment for learning interactive programming," in ACM SIGCSE, 2014, 671–676.
[10] R. G. Farrell, J. R. Anderson, and B. J. Reiser, "An Interactive Computer-Based Tutor for LISP.," in AAAI, 1984, 106–109.
[11] B. L. F. Daku and K. Jeffrey, "An interactive computer-based tutorial for MATLAB," in IEEE FIE, 2000.
[12] K. M. Y. Law, V. C. S. Lee, and Y.-T. Yu, "Learning motivation in e-learning facilitated computer programming courses," Computers & Education, 2010, 55:1, 218–228. https://doi.org/10.1016/j.compedu.2010.01.007
[13] B. B. Morrison and B. DiSalvo, "Khan academy gamifies computer science," in ACM SIGCSE, 2014, pp. 39–44. https://doi.org/10.1145/2538862.2538946
[14] R. Murphy, L. Gallagher, A. E. Krumm, J. Mislevy, and A. Hafter, "Research on the use of Khan Academy in schools: Research brief," 2014.
[15] C. Alario-Hoyos et al., "Interactive activities: the key to learning programming with MOOCs," European Stakeholder Summit on Experiences and Best Practices in and Around MOOCs, EMOOCS, 2016, 319.
[16] D. Pritchard and T. Vasiga, "CS circles: an in-browser python course for beginners," in ACM SIGCSE, 2013, 591–596. https://doi.org/10.1145/2445196.2445370
[17] B. J. Reiser, J. R. Anderson, and R. G. Farrell, "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming.," in IJCAI, 1985, 85:1, 8–14.
[18] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "Towards practical programming exercises and automated assessment in Massive Open Online Courses," in IEEE TALE, 2015, 23–30. https://doi.org/10.1109/tale.2015.7386010
[19] T. R. Liyanagunawardena, K. O. Lundqvist, L. Micallef, and S. A. Williams, "Teaching programming to beginners in a massive open online course," 2014.
[20] A. M. F. Yousef, M. A. Chatti, U. Schroeder, and M. Wosnitza, "What drives a successful MOOC? An empirical examination of criteria to assure design quality of MOOCs," in IEEE ICALT, 2014, 44–48. https://doi.org/10.1109/icalt.2014.23
[21] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," Educational Researcher, 1984, 13:6, 4–16. https://doi.org/10.3102/0013189x013006004
[22] D. C. Merrill, B. J. Reiser, M. Ranney, and J. G. Trafton, "Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems," Journal of the Learning Sciences, 1992, 2:3, 277–305. https://doi.org/10.1207/s15327809jls0203_2
[23] P. Gill, K. Stewart, E. Treasure, and B. Chadwick, "Methods of data collection in qualitative research: interviews and focus groups," British Dental Journal, 2008, 204:6, 291. https://doi.org/10.1038/bdj.2008.192
[24] Z. A. A. Muhisn, M. Ahmad, M. Omar, and S. A. Muhisn, "The Impact of Socialization on Collaborative Learning Method in E-Learning Management System (eLMS)," International Journal of Emerging Technologies in Learning, 2019, 14:20, 137–148. https://doi.org/10.3991/ijet.v14i20.10992

[25] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," Computer Science Education, 2003, 13:2, 137–172. https://doi.org/10.1076/csed.13.2.137.14200
[26] A. Pears et al., "A survey of literature on the teaching of introductory programming," in ACM SIGCSE Bulletin, 2007, 39:4, 204–223. https://doi.org/10.1145/1345375.1345441
[27] P.-H. Tan, C.-Y. Ting, and S.-W. Ling, "Learning difficulties in programming courses: undergraduates' perspective and perception," in ICCTD, 2009, 42–46. https://doi.org/10.1109/icctd.2009.188
[28] B. Means, Y. Toyama, R. Murphy, M. Bakia, and K. Jones, "Evaluation of evidence-based practices in online learning: A meta-analysis and review of online learning studies," 2009.
[29] J. Warren, S. Rixner, J. Greiner, and S. Wong, "Facilitating human interaction in an online programming course," in ACM SIGCSE, 2014, 665–670. https://doi.org/10.1145/2538862.2538893
[30] N. T. Heffernan and K. R. Koedinger, "An intelligent tutoring system incorporating a model of an experienced human tutor," in International Conference on Intelligent Tutoring Systems, 2002, 596–608. https://doi.org/10.1007/3-540-47987-2_61
[31] O. Deperlioglu and U. Kose, "The effectiveness and experiences of blended learning approaches to computer programming education," Computer Applications in Engineering Education, 2013, 21:2, 328–342. https://doi.org/10.1002/cae.20476
[32] M. Beyyoudh, M. K. Idrissi, and S. Bennani, "Towards a New Generation of Intelligent Tutoring Systems," International Journal of Emerging Technologies in Learning, 2019, 14:14, 105–121. https://doi.org/10.3991/ijet.v14i14.10664
[33] H. J. Rubin and I. S. Rubin, Qualitative interviewing: The art of hearing data. Sage, 2011.
[34] R. S. Weiss, Learning from strangers: The art and method of qualitative interview studies. Simon and Schuster, 1995.
[35] L. A. Goodman, "Snowball sampling," The Annals of Mathematical Statistics, 1961, 148–170.
[36] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement," Sociological Methods & Research, 2013, 42:3, 294–320. https://doi.org/10.1177/0049124113500475
[37] D. Armstrong, A. Gosling, J. Weinman, and T. Marteau, "The place of inter-rater reliability in qualitative research: an empirical study," Sociology, 1997, 31:3, 597–606. https://doi.org/10.1177/0038038597031003015
[38] K. A. Hallgren, "Computing inter-rater reliability for observational data: an overview and tutorial," Tutorials in Quantitative Methods for Psychology, 2012, 8:1, 23. https://doi.org/10.20982/tqmp.08.1.p023
[39] A. Yan, M. J. Lee, and A. J. Ko, "Predicting abandonment in online coding tutorials," in IEEE VL/HCC, 2017, 191–199. https://doi.org/10.1109/vlhcc.2017.8103467
[40] M. J. Lee, "(Re)Engaging Novice Online Learners in an Educational Programming Game," Journal of Computing Sciences in Colleges, 35:8, 2020.
[41] B. Tucker, "The flipped classroom," Education Next, 2012, 12:1, 82–83.
[42] K. Malterud, V. D. Siersma, and A. D. Guassora, "Sample size in qualitative interview studies: guided by information power," Qualitative Health Research, 2016, 26:13, 1753–1760. https://doi.org/10.1177/1049732315617444

10 Authors

Ruiqi Shen is a Ph.D. candidate in the Informatics Department at the New Jersey Institute of Technology (NJIT), New Jersey, USA. Her area of focus is in Computing Education.

Donghee Yvette Wohn is an Assistant Professor in the Informatics Department at the NJIT, where she directs the Social Interaction Lab. Her research area is Human-Computer Interaction in the context of social media, focusing on non-conscious use of technology, such as media habits, and their relation to psychological well-being and interpersonal relationships.

Michael J. Lee is Lee is the Dorman-Bloom Assistant Professor of Informatics at the New Jersey Institute of Technology (NJIT), where he directs the Gidget Lab. His research is in the area of Computing Education and Human-Computer Interaction, where he mainly focuses on how to improve novices' engagement and knowledge in learning programming.

Article submitted 2019-11-20. Resubmitted 2020-01-07. Final acceptance 2020-01-13. Final version published as submitted by the authors.