Comparison of Learning Programming between Interactive Computer Tutors and Human Teachers

Ruiqi Shen, Donghee Yvette Wohn, Michael J. Lee
New Jersey Institute of Technology
Newark, New Jersey, USA
{rs858,donghee.y.wohn,mjlee}@njit.edu

ABSTRACT

People typically learn programming from teachers in in-person courses or online tutorials. Interactive computer tutors—systems that deliver learning content interactively—have become more prevalent in online settings for teaching skills such as computer programming. Research has shown the efficiency and effectiveness of learning programming from teachers, interactive computer tutors, and a combination of both. However, there is limited understanding of learners’ comparative perspectives about their experience learning from these different resources. We conducted an exploratory study using semi-structured interviews, recruiting 20 participants that had experience learning programming from both teachers and interactive computer tutors. We identified factors that learners like and dislike from both learning methods and discussed the strengths and weaknesses of them. Based on our findings, we propose suggestions for designers of interactive computer tutors, and for programming educators.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**.

KEYWORDS

Tutors; interactive computing tutors; student perspectives; computing education; teachers

ACM Reference Format:
Ruiqi Shen, Donghee Yvette Wohn, Michael J. Lee. 2019. Comparison of Learning Programming between Interactive Computer Tutors and Human Teachers. In *ACM Global Computing Education Conference 2019 (CompEd ’19), May 17–19, 2019, Chengdu, Sichuan, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3300115.3309506

1 INTRODUCTION

Learning programming is considered a difficult process that requires considerable practice just like learning a natural language. However, unlike natural languages that can be used in multiple contexts and modalities in everyday experience, learners typically program within the constraints of a computer screen [26]. The difficult nature of programming may contribute to the high dropout rates in related courses in both classrooms [18, 40] and massive open online courses (MOOCs) [28, 37]. Due to this difficulty, choosing the right learning tools and methods is important for programming learners, regardless of their prior experience.

Like all learning methods, however, each has benefits and drawbacks. One-on-one tutoring would be an ideal strategy to teach and learn programming—human tutoring is one of the most effective ways for students to overcome programming barriers [15], but the lack of teachers in computing education has always been a concern of researchers and educators [16], and many learners have limited access to tailored, in-person programming courses. Even for those who have access to courses, large lecture-based formats make individualized attention from the instructor for tailored instruction unlikely [2]. Besides traditional in-classroom lectures, MOOCs are an attractive, alternative educational resource for learners since they are more affordable, accessible, and can support more students simultaneously than traditional classrooms [36, 43]. However, the lack of interactions with instructors [31] and the lack of extrinsic motivations [10, 16] are major limitations for MOOCs and are development opportunities for the future.

For the purposes of this paper, we focus only on MOOCs that deliver instruction through a virtual agent, or *interactive computer tutor* (as opposed to those primarily giving instruction through text or video). Farrell et al. were the first to introduce an interactive computer tutor (ICT) for teaching programming [9]. They described it as a system with two components: a "problem solver" (which can interpret learners’ code and give feedback), and an "advisor" (which provides tutorials that guide learners throughout the whole learning process) [9]. Systems for teaching programming such as Codecademy, Datacamp, and Treehouse, are much like the ICT described by Farrell, including a problem solver and advisor. We define ICTs as having these two features, and examine these types of systems in this paper. We choose to study ICTs rather than other types of MOOCs because literature has shown that these interactive tools are effective in delivering programming courses [7, 19] and are gaining more popularity with learners [25, 27].

While learning programming from either ICTs or teachers have shown positive student learning outcomes, there are few studies examining learners’ perspectives on their experience using and comparing these two methods. Exploring these perspectives can lead to important insights for better designs and highlight effective techniques that learners look for while learning to program.

This paper describes an exploratory qualitative study examining the learning experience from the *learners’ perspective*, contrasting their views on learning from an ICT and from a classroom teacher. By understanding the learners’ perspective on instructors, we aim to find ways to improve current ICTs' and teachers' teaching methods, and to further improve learners' educational experience.

2 RELATED WORK

2.1 Learning from Interactive Computer Tutors

The key to successfully master programming skills is through extensive practice and making mistakes [1]. However, during the learning process, students have more difficulties in applying programming in practice than understanding basic programming concepts [35]. Many MOOC websites such as Codecademy and Khan Academy provide interactive learning environments for programming. They integrate tutorials with extensive exercises and code editors with feedback systems [30]. Empirical studies have found that students learning programming interactively through well-designed computer systems achieve good learning outcomes and increased self-efficacy [9, 19].

What are the good features of an ICT for delivering programming courses? Prior work has found that watching videos and solving multiple choice questions are not sufficient in learning hands-on skills or practical programming tasks [34], and that instant feedback for students' submissions are important in learning programming [20, 21, 34]. 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) [34]. Pritchard & Vasiga summarized that embedded coding environments are beneficial for students' continuity in learning-by-doing [30]. 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 learner's community itself can act as a mentor and could possibly mitigate the issue of not having enough teachers for students [22]. Users can also recognize the benefits of features such as intentional instructional design (well-designed instructions), learning analytics (information for self-reflection), and instant feedback [42]. Our study expands on these works, aiming to explore whether these commercial systems (e.g., 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 computer tutors, which have a relatively short history in education, human teachers have been a part of education for centuries. Several studies have demonstrated the effectiveness of human teachers [4, 6, 24]. A teacher can guide students and can have them do as much thinking as possible before giving hints or answers [11]. Teachers can also intervene at the right moment to prevent students from becoming too frustrated [24], which is especially important in the early stage of learning, when learners have a higher likelihood of quitting [40, 41]. Teachers are also adaptable, with research showing that interventions in teaching styles can drastically improve teachers' effectiveness [38].

In addition to one-to-one teaching, there is also research about one-to-many teaching. Robins et al. [32] concluded that an effective programming class should stimulate students' interest and involvement, by setting clear goals and actively engaging participants in course materials and problem-solving activities [32]. However, what teaching approach contributes to an effective programming class remains a question. Pears et al.'s overview of programming classes found little systematic evidence to support any particular teaching approach that answers the question of how to teach programming effectively [29]. From a learner's perspective, Tan et al.'s survey study indicates that programming learners find the practical application of programming most difficult, and therefore lab sessions with consultation are considered more helpful than pure lectures [35]. 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 Learning From the Learners' Perspective

Based on our literature review, we found a gap in knowledge examining the learners' perspectives on receiving instruction from either ICTs or teachers. It is important to examine the learners' perspective on the difference(s) between computers and teachers, and what their preference are when interacting with either of these choices when learning to program. The learning method they choose may influence their perception of programming skills, and therefore, potentially affect retention rates and learning experiences. Therefore, we explore the following two research questions in this paper:

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

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

3 METHOD

To answer these research questions, we conducted in-person, semi-structured interviews with 20 subjects. Interviews are a commonly used method for exploratory studies [8, 33, 39], especially to gain an in-depth perspective into subjects' views and experiences when there is limited research in the literature [13]. We recruited participants through snowball sampling [14], where each participant suggested at least one additional person that met our inclusion criteria and that they thought would be a good candidate for us to interview. The initial six participants were students from a mailing list that represented a wide range of demographics (e.g., gender, ethnicity, age, job/major) from two public universities in the northeast United States (US), with subsequent participants being classmates, alumna, or professional colleagues distributed across the US.

One researcher conducted all the interviews, either in-person (n=16) or on the phone (n=4). All interviews were recorded, averaging 29 minutes per interview. We required participants to have 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 [9]). We intentionally did not specify ICTs any further, so that our participants could talk broadly about the technologies they had used without being limited to a specific type of ICT.

The interview questions included two major 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 coded the interview transcripts 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 [5]. 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) [5]. 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). During the process, we read through those tagged texts, and consolidated similar tags into one tag (i.e., code families [5]), or split one tag into different tags. This resulted in 19 themes (each research question has several themes) and more than 50 tags. We analyzed all 20 interview transcripts, reaching a high level of intercoder reliability (.87) and intercoder agreement (.92)[1]. We present representative quotes from participants in our results 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 experience in programming ranged from 1 to 15 years (median 4.5).

4.1 RQ1a: What Do Learners *Like* About Learning Programming from Interactive Computer Tutors?

4.1.1 Provides a Code Editor. A code editor allows learners to write and run their code directly on the ICT. Most of the participants considered this feature to be helpful, and there are three main reasons: First, a code editor dismisses the need to set up a local environment. 7 out of 20 participants mentioned that they did not need to set up local environment when they only wanted to learn some basics. The code editor saves time from having to set up a local programming/development environments.

Second, a code editor provides one-window convenience. Participants mentioned that they liked the code editor because they could see the tutorials, examples, and do the exercises in the same window, which was more convenient than switching windows between tutorials and coding tools. P9 told us how she found the code editor 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 code editor because it was similar to the real coding environment but better in terms that the code editor in the ICT could give customized feedback while a real environment could not.

[1]Scores were calculated using the proportion agreement procedure [5]. 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 [3, 17].

4.1.2 Content Design. This refers to the course materials and the way ICTs organize and deliver information. The content design can be grouped into three main features. First, ICTs organize and display lessons with a clear outline, which learners can use to see exactly where they are in the learning process (i.e., curriculum). P20 described how she found the organization useful in Dataquest: "*The lessons are very simplified [in Dataquest]. They are broken down into different modules, so it makes it very easy to consume.*"

Second, ICTs provide practice immediately after each tutorial, so learners can (re)apply whatever they learned quickly. P3 mentioned how the 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, ICTs provide examples. Participants mentioned that the examples from ICTs were very helpful to understand the lessons. P5 is a novice programmer, and his biggest concern was that he could not visualize what his code would output. He described his experience of learning web development in Codecademy, and how it helped him with examples: "*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.*"

4.1.3 Flexible. Participants enjoyed the flexibility in learning when using ICTs. Flexibility allows learners to go at their own pace whenever and wherever they want. 8 participants mentioned that they wished to learn at their own pace, and so learning from ICTs could satisfy their needs. P2 told us that he preferred online learning because he could learn at his own pace: "*For 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.*"

With an ICT, learners can learn whenever and wherever they want. P20 gave us her opinion on location flexibility: "*Like, I don't have Python installed on my phone and I can still do my lessons [on Dataquest] maybe 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.

4.1.4 Efficient. Some participants compared the ICT with other resources and concluded that they liked the efficiency when learning from ICTs. 2 participants compared the time spent learning from an ICT and from a teacher in a classroom. They felt that being present in a physical classroom 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.*"

Four participants compared ICTs with textbooks. They 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 for me 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.*"

4.1.5 Provide Sufficient Help. Participants mentioned that they could use in-context resources within the interface to get assistance if needed. There are three specific features that provide sufficient help for learners: (1) hint systems, (2) staff help, and (3) discussion panels. A hint system is the basic feature providing help (usually generated automatically), and the very first help learners will 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.

If the hint system fails to achieve learners' expectations, some ICTs provide staff help. Staff are real people who are 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.*"

Built-in discussion panels also provide a place where learners can discuss their questions and ask for help. P9 liked discussion panels very much and she said: "*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.*"

4.1.6 Designed for Various Learner Levels. Some ICTs provide different pathways based on skill-level. With these, learners can find courses matching their experience. P16 told us: *"For Codecademy, they have levels, like 'did you just start learning code,' 'you already have some experience,' 'you're an expert,' like that. So that really helps because if you already know coding you don't need a very simple example because it's too easy.*"

Interestingly, one participant mentioned that he liked the feature of Codecademy that locks access to later content until finishing 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. Compared with a long video tutorial, these short videos relieve the cognitive load of learners.

4.2 RQ1b: What Do Learners *Dislike* About Learning Programming from Interactive Computer Tutors?

4.2.1 Content Design. This again refers to the course materials and the way ICTs organize and deliver information. While most people liked the content provided by ICTs, there were also participants who did not like the content design. Four participants thought that the tutorials and practice materials were too basic to be useful. They liked ICTs but wished they could provide more advanced content. P2 had 4 years of programming experience; he considered himself as having a good understanding of programming basics, and expressed his concerns: "*I was doing C++, 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.*"

One participant disliked course content because the sections were redundant. He said: "*At the beginning, I will 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.*"

While some experienced learners considered the content covered by ICTs to be too basic, other junior-level learners mentioned that sometimes, information were too brief to understand. Some ICTs provide short introductions, but do not really explain the logic behind the material. P11 started programming 2 years ago, and was struggling to understand the complex logic behind 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.

4.2.2 Locks Access to More Advanced Concepts. Participants said that some ICTs required them to finish the current module before unlocking the next one. While we mentioned that one participant liked this feature earlier, four participants disliked this feature. They had prior experience and had clear goals on what they had to learn. This feature limited their learning efficiency. P13 was a working professional who had 2+ years of programming experience and learned programming for fun in his leisure time. He said: "*I 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.*"

4.2.3 Does not Provide Sufficient Help. Although some participants believed they could get sufficient help from ICTs, four participants disagreed. These participants did not believe that the tutor could guide them effectively to figure out the logic behind a problem. P5, a novice programmer, shared his experience getting stuck on a problem in Codecademy: "*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?

4.3.1 Has Real Life Experience in Programming. 7 out of 20 participants mentioned that they liked to learn from teachers who share their real-life experience in programming. These experiences include: how to avoid common mistakes, how to style the 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.*" P11 had 2 years of experience in programming, but she 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 and can focus on parts that students have the most difficulty understanding.

4.3.2 Provides Solid Learning Experience. Participants believed they could learn programming more concretely and systematically with teachers. Teachers introduce new concepts, but also provide more information and background about these concepts. Teachers also help students stay on the right track. P4 had 10 years of programming experience, and 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.*"

4.3.3 Has Conversations. Conversations with teachers are valuable to learners. Participants thought that they could discuss ideas and explain their problems better through conversations. P17 was experienced in programming and was full of project ideas that he would like 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 he is experienced, so he'll give you some feedback on your ideas and you know how to improve yourself or how to improve your program.*"

Another 3 participants felt conversations allowed teachers to better understand students' problems. Face-to-face conversation with experts is an easier way to get problems solved than sending emails or searching for answers elsewhere, because learners can use various methods to express themselves in-person (e.g., drawing, writing, gesticulating). P10 told us that she could show her work directly to teachers when communicating face-to-face, so that the teachers can better understand her questions with real examples.

4.3.4 Provides Real-Time Help. Participants mentioned that they could have their questions answered immediately when they were in class with teachers. When learners program, it is common to encounter errors. However, if these errors are not solved immediately, it may lead to other issues, causing the learner to get stuck. ICTs usually cannot provide real-time help for specific questions, while teachers can. When asked why they like to learn programming from teachers, P11, a beginner said: "*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: "*So, in-person it's more instant and I can do some things right away and get out the way whatever question I have.*"

4.3.5 Displays Code Example. Three participants liked teachers who showed code examples in class. P2 had 4 years of programming experience; he had some teachers who would just lecture for hours and teachers who showed code examples. He thought that he got little from the former because he could not understand the lecture content without any examples. He had a teacher who made class easier by showing code examples. He said: "*She basically taught and then she had her code that helped us, whereas other professors were just lecturing for three hours.*"

4.4 RQ2b: What Do Learners *Dislike* About Learning Programming from Teachers?

4.4.1 Not Efficient. Most participants thought that learning from teachers was not efficient. Teachers typically take time in providing assignments and giving feedback. In contrast, 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 lecturing style of the teacher may also be less efficient than reading the course materials. P6 had 15 years of programming experience. Most of the teachers he had would just read from the textbooks. He expressed his frustration with these types of teachers since he could just read from the textbook himself at home.

In addition, three participants felt online materials were easier to access than teachers. For example, P13 stated that online materials could be accessed quickly, while for teachers, he needed to register and pay for a course, and even be physically present in class.

4.4.2 Does Not Have Same Pace with Students. Teachers' speed of instruction was also a large concern for learners. 9 out of 20 participants had the experience of being unable to keep up with the teachers' pace. This happened when the teacher delivered the content too quickly, or when students had difficulties understanding some points, but the teacher kept moving forward. For example, P20 had a fast-paced programming course. As a novice programmer with little experience, she could not catch up with the progress, so she turned to online courses for help. She said: "*And with class, things go by so quickly, we meet 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 too slow. P16 was a student who always learned things quickly, and so, when she took a class but was ahead of the teacher's pace, she felt bored. 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.*"

4.4.3 Provides Inflexible Curriculum. Sometimes, the content provided in a course were not what learners wanted. P12 had learned programming for years. When he was in college, he selected a C++ course, wishing to learn some advanced topics. However, the course he attended only covered 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 wished to learn some basics, but what the teacher taught was more advanced. She said: "*I figured I will get a tutor to teach me the basics, because he [professor] didn't teach us the basics.*"

4.4.4 Teaching Competency. Some of the participants questioned their teachers' programming competence. Three participants thought their teachers were not experienced in programming, and believed that a good programming teacher should be a good programmer. P9 was disappointed with her programming teacher when he could not explain example code in detail. She said: "*he is not experienced in programming. All of his code is just copy-paste from another website, and then sharing it to you. He can't explain any details to you.*"

Programming skills and new technologies are constantly changing, but participants had concerns that their teachers' skills were not up-to-date. For example, P12 had a solid foundation in programming; his goal was to learn new technologies. He found that teachers in school did not satisfy his needs. "*I think the teacher is usually far behind the current progress [...] But what I want to learn, is always something new,*" he said.

4.4.5 Is Not Responsive. Some participants felt that they lacked teacher attention in large classes. "*They [professors] are always busy; your problems might not be solved in time,*" P18 said.

Lack of responsiveness may also be attributed to a teacher's personal style. P17 told us one of his teachers who never replied their 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.*"

5 DISCUSSION

We identified many features that make learners like or dislike learning from ICTs and teachers. Efficiency and practice were the two main factors that learners care about. Since most of our participants learned programming with the goal of applying it quickly into work or study, learning efficiency was their biggest concern. Most of the participants thought that learning-by-doing was the best way to master programming skills, so immediate practice was also a consideration when choosing learning methods. In addition, our participants were not satisfied with single-media instructions such as textbooks or video. Our findings have implications for designers of ICTs and teachers who teach programming.

For ICTs, the biggest strengths that most participants mentioned were code editors and additional practice, while a major weakness was the content design (too basic, repetitive, or brief). To address this issue, designers could provide more advanced, practice-oriented tutorials by taking advantage of the availability of code editors.

The existence of both basic and advanced experience levels leads to another problem we identified during interviews. Two participants liked the features that some ICTs separate the content for different level of learners; one beginner programmer liked the features that the computer 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 were good at delivering content with related exercises in an efficient manner, while teachers did a better job in providing customized help with real life experience. Both ICTs and teachers can gain benefits from each other. First, teaching applications can hire experts, who can provide help for questions when requested by online learners. Experts can also be present in the system's online learning community to have conversations with learners.

Second, one of our findings suggests that teachers were experienced in delivering knowledge, so they knew what parts of the course content might be most difficult to students, so that they could pay extra attention when teaching those parts. ICTs can achieve this feature by gathering data about different sections of their course content (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, most participants indicated that they valued the opportunity to have conversations with teachers. Reasons include gaining coding tips, real life experience as programmers, exchanging project ideas, and getting help with questions. Listening to long lectures without interactions were disliked by students. There is potential to combine ICTs into programming classes to compliment teachers. Our findings suggest that existing ICTs do a good job in delivering basic concepts and exercises. Therefore, teachers may be able to have ICTs deliver basic information, and spend the time saved having more conversations with students regarding problems, projects, and real-life experience in programming.

6 LIMITATIONS & FUTURE WORK

Our study has limitations that present opportunities for further research. First, our recruitment method may have introduced a sampling bias. However, we found that our participants represented a wide variation of demographic factors and years of experience with coding. Second, we had a total of 20 participants in our study, which may raise questions about the representativeness of our sample and generalizability of our findings. We reached data saturation [12, 23] on our 16th interview and verified that our additional participants did not provide substantially different information from prior participants. Third, we identified that factors such as learning environment (e.g., summer camp, college course, vocational training) and learning objectives (e.g., learning for work, school practice, or personal interest) may affect how learners evaluate their learning experience. We will conduct further research to explore whether learning environments and learning objectives, or even other factors (e.g., gender, age, job, level of experience, order of learning from a specific type of tutor), affect how learners evaluate their learning experience(s). Lastly, we used participants' self-reported number of years in programming to describe them in our study (e.g., "P11 had 2 years of experience in programming."). However, self-reported years of experience may not reflect participants' actual programming ability or expertise. Future studies can examine the relationship between years of experience and programming ability. 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 instructions 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 strength and weakness of learning from interactive computer tutors and teachers, which we use as a basis for design suggestions for these types of instruction.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grant IIS-1657160. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

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

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

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

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

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

[6] Peter A Cohen, James A Kulik, and Chen-Lin C Kulik. 1982. Educational outcomes of tutoring: A meta-analysis of findings. *American Educational Research Journal* 19, 2 (1982), 237–248.

[7] Brian LF Daku and Keith Jeffrey. 2000. An interactive computer-based tutorial for MATLAB. In *Frontiers in Education Conference (FIE)*, Vol. 2. IEEE, F2D–2.

[8] Kathleen DeMarrais. 2004. Qualitative interview studies: Learning through experience. *Foundations for research: Methods of inquiry in education and the social sciences* 1, 1 (2004), 51–68.

[9] Robert G Farrell, John R Anderson, and Brian J Reiser. 1984. An Interactive Computer-Based Tutor for LISP.. In *AAAI*. 106–109.

[10] Gerhard Fischer. 2014. Beyond hype and underestimation: identifying research challenges for the future of MOOCs. *Distance Education* 35, 2 (2014), 149–158.

[11] Barbara A Fox. 1991. Cognitive and interactional aspects of correction in tutoring. *Teaching knowledge and intelligent tutoring* 01 (1991).

[12] Jill J Francis, Marie Johnston, Clare Robertson, Liz Glidewell, Vikki Entwistle, Martin P Eccles, and Jeremy M Grimshaw. 2010. What is an adequate sample size? Operationalising data saturation for theory-based interview studies. *Psychology and Health* 25, 10 (2010), 1229–1245.

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

[14] Leo A Goodman. 1961. Snowball sampling. *The Annals of Mathematical Statistics* (1961), 148–170.

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

[16] Mark Guzdial. 2014. Limitations of MOOCs for Computing Education-Addressing our needs: MOOCs and technology to advance learning and learning research (Ubiquity symposium). *Ubiquity* 2014, July (2014), 1.

[17] Kevin A Hallgren. 2012. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology* 8, 1 (2012), 23.

[18] Päivi Kinnunen and Lauri Malmi. 2006. Why students drop out CS1 course?. In *ACM International Computing Education Research*. ACM, 97–108.

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

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

[21] Michael J Lee, Andrew J Ko, and Irwin Kwan. 2013. In-game assessments increase novice programmers' engagement and level completion speed. In *ACM Conference on International Computing Education Research*. ACM, 153–160.

[22] Tharindu R. Liyanagunawardena, Karsten O. Lundqvist, Luke Micallef, and Shirley A. Williams. 2014. Teaching programming to beginners in a massive open online course. (2014).

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

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

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

[26] Robert Moser. 1997. 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*, Vol. 29. ACM, 114–116.

[27] Robert Murphy, Larry Gallagher, Andrew Krumm, Jessica Mislevy, and Amy Hafter. 2014. Research on the use of Khan Academy in schools: Research brief. (2014).

[28] Daniel Fo Onah, Jane Sinclair, and Russell Boyatt. 2014. Dropout rates of massive open online courses: behavioural patterns. *EDULEARN* (2014), 5825–5834.

[29] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A survey of literature on the teaching of introductory programming. In *ACM SIGCSE Bulletin*, Vol. 39. ACM, 204–223.

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

[31] Liana Razmerita, Kathrin Kirchner, Kai Hockerts, and Chee-Wee Tan. 2018. Towards a Model of Collaborative Intention: An Empirical Investigation of a Massive Online Open Course (MOOC). In *Hawaii International Conference on System Sciences*.

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

[33] Herbert J Rubin and Irene S Rubin. 2011. *Qualitative interviewing: The art of hearing data*. Sage.

[34] Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, and Christoph Meinel. 2015. Towards practical programming exercises and automated assessment in Massive Open Online Courses. In *Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, 23–30.

[35] Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. 2009. Learning difficulties in programming courses: undergraduates' perspective and perception. In *Computer Technology and Development*, Vol. 1. IEEE, 42–46.

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

[37] Colin Taylor, Kalyan Veeramachaneni, and Una-May O'Reilly. 2014. Likely to stop? predicting stopout in massive open online courses. *arXiv preprint arXiv:1408.3382* (2014).

[38] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. 2014. A systematic review of approaches for teaching introductory programming and their influence on success. In *Conference on International Computing Education Research*. ACM, 19–26.

[39] Robert S Weiss. 1995. *Learning from strangers: The art and method of qualitative interview studies*. Simon and Schuster.

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

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

[42] Ahmed Mohamed Fahmy Yousef, Mohamed Amine Chatti, Ulrik Schroeder, and Marold Wosnitza. 2014. What drives a successful MOOC? An empirical examination of criteria to assure design quality of MOOCs. In *Advanced Learning Technologies (ICALT)*. IEEE, 44–48.

[43] Li Yuan, Stephen Powell, JISC CETIS, and others. 2013. MOOCs and open education: Implications for higher education. (2013).