# Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers

**April Y. Wang[1], Ryan Mitts[1], Philip J. Guo[2], and Parmit K. Chilana[1]**

[1]Computing Science
Simon Fraser University
Burnaby, BC Canada
{ayw7, rmitts, pchilana}@sfu.ca

[2]Design Lab
UC San Diego
La Jolla, CA USA
pg@ucsd.edu

## ABSTRACT

Conversational programmers represent a class of learners who are not required to write any code, yet try to learn programming to improve their participation in technical conversations. We carried out interviews with 23 conversational programmers to better understand the challenges they face in technical conversations, what resources they choose to learn programming, how they perceive the learning process, and to what extent learning programming actually helps them. Among our key findings, we found that conversational programmers often did not know where to even begin the learning process and ended up using formal and informal learning resources that focus largely on programming syntax and logic. However, since the end goal of conversational programmers was not to build artifacts, modern learning resources usually failed these learners in their pursuits of improving their technical conversations. Our findings point to design opportunities in HCI to invent learner-centered approaches that address the needs of conversational programmers and help them establish common ground in technical conversations.

### Author Keywords

Conversational programmers; learner-centered design; programming literacy; technical conversations.

### ACM Classification Keywords

K.3.2 Computers and Education: Computer and Information Science Education—*literacy, computer science education*.

## INTRODUCTION

Considerable research efforts have been devoted to human-computer interaction (HCI) and computing education research towards lowering the barriers to learning programming. Many of these efforts have contributed innovative tools and approaches to support the programming needs of a variety of learners, such as computer science (CS) students [17,28,53], end-user programmers [14,15,30,32] and professional programmers [1,3,13]. A large focus of these projects has been on improving learners' understanding of programming syntax and logic and facilitating interaction with feature-rich programming environments as these are known to present key challenges for new learners.

Unfortunately, most of what we know about the programming learning process and the challenges that learners face is based on studies of students in the classroom [53] or professionals in industry [1]. Only recently have we started seeing studies into informal learning processes among non-traditional populations, such as designers [15], high school teachers [43], and older adults [22]. Given this increased diversity in learning needs and the backgrounds and skills of programming learners, there have been increased calls [24] to better understand the goals of such diverse learners and their interaction with modern learning resources.

Pushing on this idea of learner diversity, recent work suggests that there is a unique class of learners who are motivated to learn programming, but never actually need to write code [7,8]. These learners are termed as *conversational programmers* as they seek to acquire programming skills only to engage more effectively in technical conversations or to improve their job marketability (e.g., in marketing, sales, UI design, or management). Although prior work has established the existence of such a population of conversational programmers at a single technology company [8] and in the classroom [7], do such people exist more broadly in other more diverse settings and similarly learn programming to improve technical conversations? Several other questions also remain unanswered: how do conversational programmers actually approach learning programming when their goal is not to write code? To what extent are their learning approaches similar or different from other non-traditional learners, such as end-user programmers? And, do conversational programmers even find it useful to learn programming to improve their technical conversations?

In this paper, inspired by the idea of learner-centered design [24,51], we investigate the learning needs and strategies of conversational programmers. We took a qualitative approach for this investigation and recruited a broad range of people representing different professions in local companies and educational and non-profit institutions (e.g., archivist,



*CHI 2018,* April 21–26, 2018, Montreal, QC, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5620-6/18/04…$15.00
https://doi.org/10.1145/3173574.3174085

artist, entrepreneur, HR coordinator, admin staff, psychologist, event manager, marketing assistant, medical instructor and visual designer). We carried out 23 interviews (14 female) with a diverse set of participants who did not have a formal degree in CS, did not work in an engineering role, and were not required to write code on the job, but had tried to learn programming. Our interviews focused on uncovering the kinds of challenges these conversational programmers faced in technical conversations and how and why they made use of different approaches and modern resources for learning programming. The interviews also probed into the participants' perceptions of whether or not their efforts in learning programming were actually helpful for their conversations or other aspects of their jobs.

Our key findings illustrate a variety of challenges and misunderstandings that conversational programmers can encounter in technical conversations and that can eventually motivate them to explore programming. However, we found that most conversational programmers often do not know where to even begin the learning process and typically seek recommendations from other programmers or rely on popular web search results. This leads them to invest in formal and informal learning strategies that are typically designed for professional or end-user programmers and heavily focus on syntax and logic issues in code. However, since the end goal of conversational programmers is not to build artifacts, a mismatch ensues between their expectations and what these learning resources offer, with conversational programmers often feeling like they have failed.

The main contribution of this paper is in providing empirical evidence characterizing the unique learning needs of conversational programmers, how these needs differ from populations of end-user programmers and professional programmers, and how modern learning resources that focus on artifact-creation can fail conversational programmers.

## RELATED WORK

Our study builds upon prior work in HCI and computing education that focuses on non-traditional learner populations (e.g., learners who are not CS majors or professional programmers) and how people interact with formal and informal programming learning environments.

### Studies of non-traditional programmers

End-user programmers were among the first group of non-traditional programmers to receive attention in the literature. This class of programmers consists of people who write code not for professional software development tasks, but to solve a domain-specific problem or to improve their productivity in a particular domain [32]. It is estimated that the population of end-user programmers is much larger than professional programmers [47], and many studies have been carried out to understand why and how different groups of end-user programmers learn programming. For example, web designers and data scientists write scripts for domain-specific project needs, and they mainly learn by "head-first" and "trial and error" methodologies [13,15,28] often by consulting books, code examples, blogs, and forums [14,15].

Recent studies show that another emerging non-traditional learner population consists of conversational programmers [7,8]. Past surveys indicate that this population is mainly motivated to learn programming to improve the efficacy of technical conversations and to acquire marketable skillsets. Although there was some indication that conversational programmers at a large technology company were using online resources, courses, books, and help from colleagues to acquire programming skills, prior work does not provide any insights into the actual learning strategies and approaches used by these learners, and whether they actually succeeded in improving their technical conversations. Our work adds insights into how conversational programmers exist in diverse job sectors, how and why they use different learning resources, and how they perceive those available resources.

K-12 teachers tasked to teach CS are another group of people who learn programming on-the-job [43,44], and they share some similarities with conversational programmers. Although teachers may never need to write code on-the-job [43], they still need to understand programming syntax and logic since they need to teach those in class, grade coding assignments, and answer coding-related questions. There is some indication that these teachers can have feelings of isolation in the learning process and may benefit from having their own dedicated learning communities. Our study found similar sentiments amongst conversational programmers.

### Formal learning environments for programming

Formal learning is defined as an activity that has a structured curriculum with clearly defined objectives carried out within a defined schedule, such as a school or college course, or a workshop [52]. Research on non-CS major students taking intro CS courses [7,19,56] revealed that not everyone learning programming intends to become a professional programmer, and traditional intro CS courses failed to engage non-CS major students. With growing calls for learner-centered design [24], some recent work has explored formal ways of making programming relevant for non-CS students [19,20,23,25,40]. For example, efforts have been made to teach programming skills in the context of media computation [23,25], and introducing the concepts of natural language processing (NLP) and artificial intelligence (AI) in a non-programming context [35].

In addition to traditional K-12 and college classrooms, MOOCs (Massive Open Online Courses) for programming have become popular among some adult learners [18,59]. Other emerging formal learning environments include coding bootcamps where adults who want to improve their practical coding ability can focus on particular topics for a short period of time. Although these formal learning methods require less of a time investment than college courses, doubts have been raised about whether bootcamps or MOOCs actually work for people who seek to improve their employment prospects [29,54]. Our study further reveals that these formal approaches present cost vs. benefit tradeoffs that are even more

acute for conversational programmers, making them less popular among this population of learners.

### Informal learning resources for programming

In contrast to formal learning, informal learning consists of activities that are unstructured, self-directed, and initiated in response to some need, often on-the-job [41,52]. The learner typically self-manages this type of learning and focuses on improving certain skills or addressing specific gaps in knowledge. In terms of informal ways of learning programming, considerable attention has been paid to investigate how people can learn programming online.

For example, several studies have examined why and when online interactive coding tutorials are useful [27,31,36]. Although these tutorials can help learners with artifact-creation needs (e.g., professional or end-user programmers) get started, their utility is perceived to be limited as tutorials are rarely tailored to learners' prior coding knowledge. Our study further shows that even conversational programmers experience feelings of failure with such informal resources, but for different reasons. For example, for conversational programmers the key drawback is that these informal resources focus mostly on syntax and logic issues and provide less conceptual explanations.

Another class of research has explored informal learning and information seeking behaviors on discussion forums for novice programmers [3,38]. These forums effectively facilitate discussion and peer-to-peer knowledge exchange among learners writing code [38,46,49]. But, as discussed in prior work [17,40], we also found that the identity of the user and type of forum can affect how well users participate in these discussions. Furthermore, we found that conversational programmers often felt like "outsiders" in communities targeting artifact creation needs.

## METHOD

To study the learning strategies of conversational programmers, we conducted semi-structured interviews with 23 participants from a variety of backgrounds (Table 1).

### Participants and Recruitment

We recruited self-identified conversational programmers through personal connections and snowball sampling, advertising posters at educational organizations, and through mailing lists of local meet-up groups for programming over a 4-month period in 2017. Our participants had to fit the following criteria to take part in the interviews: 1) not have a formal degree (or even a minor) in computer science, engineering or IT; 2) not be working in any kind of a software development or engineering role or any role requiring programming on-the-job; and, 3) must have recently tried to learn programming or CS either informally or formally.

We ended up with 23 study participants (14 female) as we aimed for diversity in job roles, age, and gender. As shown in Table 1, our participants held a variety of positions (e.g., artist, psychologist, pharmacist, entrepreneur, library archivist, bank clerk, medical instructor). They also brought in different levels of experience, ranging from being an intern to a senior manager with 20 years of experience.

### The interview instrument

Before the interview, we collected basic demographic information through a questionnaire (e.g., age, gender, occupation, education and previous experiences with programming languages). We began the interview with some warm-up questions. For example, we asked them to describe their current work and recall the most recent situation in which they were required to have a technical conversation.

Next, we asked questions about their learning process and strategies, focusing on resources they used, in which situation they used those resources, how they knew where to look at resources and to what extent they found the resources to be useful. Initially we used common resources for learning programming to prompt the participants if necessary (e.g., programming courses, books, online documentation, *Stack Overflow*, MOOCs). After the first five interviews, we updated this list with additional informal resources that came up in the interviews so far (e.g., *Wikipedia*, articles, news, blogs, magazines, *YouTube videos*).

Lastly, we ended the interview by probing into conversational programmers' perceptions of the learning process, asking them to reflect on what they felt they achieved after all their learning efforts and whether (or not) they wanted to keep learning programming in the future.

### Data Analysis

We transcribed the audio recordings and did an open coding of the data using *ATLAS.ti.* We used an inductive analysis [12] approach and affinity diagrams to explore the themes around our main research questions. Three members of the research team first began with an open coding pass to individually create a list of potential codes. Upon discussion and use of affinity diagrams, a single coding scheme was devised and two team members independently coded two of the transcripts using this scheme. The first pass inter-rater reliability test achieved a Kappa score of 0.61 as there was some confusion about redundant codes and where they should be used. Upon further discussion and iteration with the research team, we revised the coding scheme, merging the potentially over-

| ID | Age | Occupation | ID | Age | Occupation | ID | Age | Occupation |
|---|---|---|---|---|---|---|---|---|
| P1 | 31-40F | entrepreneur | P9 | 19-30F | advertising manager | P17 | 41-50M | product manager |
| P2 | 19-30M | visual designer | P10 | 31-40F | health scientist | P18 | 31-40F | humanities scholar |
| P3 | 41-50F | bank clerk | P11 | 19-30F | library archivist | P19 | 19-30F | artist |
| P4 | 41-50F | HR coordinator | P12 | 19-30M | business assistant (intern) | P20 | 31-40F | marketing coordinator |
| P5 | 19-30M | helpdesk support (intern) | P13 | 19-30M | product manager | P21 | 19-30M | business assistant (intern) |
| P6 | 51-60F | pharmacist | P14 | 19-30F | HR coordinator | P22 | 51-60F | medical instructor |
| P7 | 19-30M | business development manager | P15 | 19-30F | university administrative staff | P23 | 31-40F | psychologist |
| P8 | 19-30M | marketing coordinator | P16 | 19-30M | marketing assistant (intern) | | | |

**Table 1. Our participants from local companies and educational and non-profit institutions represented a diverse range of occupations**

lapping codes and removing the infrequent codes. Next, the two raters applied the revised coding scheme on a new subset of interview transcripts, achieving a higher Kappa score of 0.87. We next used axial coding to discover relationships among emerging concepts, followed by selective coding to identify recurring themes.

### Presentation of Results

Our analysis revealed a number of themes and next we focus on presenting key results on why conversational programmers wanted to learn programming, how they approached learning programming, how they perceived and struggled in the learning process, and, paradoxically, why they still had a positive attitude towards learning programming.

## KEY REASONS FOR LEARNING PROGRAMMING

As shown in Table 1, our study participants were professionals and domain experts in a variety of roles and did not need to write code on-the-job. In their responses to motivations for learning programming, we saw many similar responses to previous studies [7,8] of conversational programmers: our participants mainly wanted to learn programming to improve their technical conversations (16/23) or to enhance their future marketability (7/23). In addition, some participants were interested in using their programming skills to perform end-user programming tasks (5/23), to gain credibility with their technical team members (4/23), and to stay current with digital trends and technology developments (4/23).

Given that a key motivation for learning programming was improving technical conversations, we first shed light on why our participants found it challenging to converse with developers and other technical personnel.

### Challenges in understanding the context of conversations

Participants commonly reported that they felt lost in understanding the full context of implementation decisions made by software developers that involved low-level details or high-level concepts, such as machine learning.

Some participants said they found it difficult to follow along and make sense of important technical conversations because they simply did not have a shared vocabulary. For example, an advertising manager described her challenge in interpreting the data that the development team collected for campaign planning:

*We do a lot of the advertising work on the internet and we have programmers who gather data for planning campaigns. I always need to contact them to figure out how they collect it. So, the conversations are very difficult... especially when they mention terminologies around network, database, big data, and algorithms... I feel like I have to learn from the beginning, and that's why I am learning Python right now. (P9)*

In other cases, conversational programmers were not only required to listen and understand the technical conversations, but also to be able to talk using technical terminology. For example, an entrepreneur from a local start-up company, who was usually invited to give keynotes on innovation strategies or investment pathways, explained how she had to make sure her understanding of certain terminology was *"100% accurate":*

*If something was wrong about a technical concept [that I learned], and then if I were to say it in front of people who are world leaders…that would be embarrassing. (P1)*

### Challenges in building rapport

In addition to better understanding the context during technical conversations, our participants were motivated to learn programming to build rapport with technical people as well. Our participants' narratives revealed how they often experienced strains in their professional relationships or felt ignored because of their lack of programming knowledge:

*…the programming people tend to be not interested in talking to me. We don't really speak the same language. (P3)*

By learning programming, some participants felt they could gain respect and credibility from their technical teams. For example, a business development manager whose job was to provide customer feedback to developers said:

*…if you can write code or you can understand code, developers respect you more…they would "let you in" …when you're having a conversation it's easier for you to get what you want. (P7)*

Another participant working in a technology consulting company found it useful to socialize with developers by better understanding and making programming-related jokes:

*Our company has a shared space as resources for other companies to use…I became close friends with a number of companies, as well as, a lot of them are our clients as well... Learning some basic syntax, I was able to joke about basic stuff like, "Man, I messed up one comma, and I've messed up my entire code!" Little jokes and nuances that people who know the language can laugh about really helps me start the conversation. (P13)*

In summary, our participants were mainly motivated to learn programming because they believed that it would help them better understand the context of technical conversations and build rapport with technical people on the other side of these conversations.

## APPROACHES USED FOR LEARNING PROGRAMMING

To investigate how conversational programmers tried learning programming, we focused on eliciting the different approaches and resources that our participants attempted to use.

### Beginning the learning process

Most of our participants (19/23) mentioned that they often did not even know where to start the learning process and their first instinct to learn programming was to ask an expert (e.g., a colleague, friend, or more technical family member):

*I think if I had a programming background, I probably would have been able to find information a lot easier and quicker, but because I had to browse through so much and I didn't understand some of the lingo…so, I found it easier just to ask my developer-colleagues like what website should I go to if I want more information on this [programming language]. (P20)*

| Formal approaches |
|---|
| **In-person courses** (e.g., night courses at community colleges) |
| **Bootcamps & workshops** (e.g., HTML bootcamp; Python one-day workshop) |
| **Online courses** (e.g., Lynda.com, Coursera, Udacity, edX) |
| **Informal approaches** |
| **Online reference resources** (e.g., W3Schools, Wikipedia, company's internal references site, specific services such as Drupal ) |
| **Forums** (e.g., Reddit, Quora, Stack Overflow, Facebook Groups) |
| **Online coding tutorials** (e.g., Codecademy, FreeCodeCamp) |
| **Popular press** (e.g., Tech Insider; Bloomberg; TechCrunch) |

**Table 2. Formal and informal resources used by participants**

In fact, participants reported that they relied on experts throughout the learning process: to confirm the relevance of what they found online, to seek definitions or clarifications of technical terms, or to help them debug the coding problems that were encountered during the learning process.

Another approach to getting started that participants described was that they would just try to search online and try to follow the top search results. Several participants described how they relied on *Google* in particular to look up programming-related definitions of terminologies:

*When I google search these terminologies, I click on Wikipedia all the time because Wikipedia pops up quite heavily in the first few search columns. (P13)*

Using guidance from an expert or suggestions from online search, our participants ended up investing in different formal and informal learning approaches described below. Overall, participants mentioned trying out 21 different programming languages (e.g., *HTML, CSS*, *JavaScript*, *Python*, *PHP*, *Ruby*, *SQL*, *R*, *VBA*) as well as finding information on over 20 different technical concepts, such as "machine learning", "big data", "cloud computing" and "blockchain".

### Formal and informal learning approaches

We summarize the key formal and informal learning approaches described by participants in Table 2. Although our participants were more likely to use informal learning resources, a few participants had invested in even paid formal methods to seek more guided instruction, such as in-person short-term college programming courses (2/23), attending bootcamps or programming workshops (7/23), and signing up for free online courses (6/23) through *Lynda.com*, *Coursera*, and *CS50* at Harvard.

Since our participants had tried many types of informal approaches, we have categorized their top responses below.

**Online reference resources:** Some participants sought information on explanations of terminology and usage of API instructions using online reference resources usually suggested in search results. Many participants (10/23) visited online documentation sites, such as coding reference sites (e.g., *W3Schools*) and service/product sites (e.g., *Amazon Web Services*). Similarly, Wikipedia was also widely used by participants (9/23), particularly for checking definitions of unfamiliar terminologies brought up in technical conversations.

**Forums**: Most of the participants (16/23) had come across online forums, such as for specific services, (e.g., *WordPress, Drupal*), coding forums (e.g., *Microsoft forums*) and general-purpose platforms (e.g., *Quora, Reddit, Facebook Groups, Slack Groups*) to seek information related to programming. However, participants were not actively involved in typical online communities for developers. For example, most of the participants (18/23) had never used or even heard of Stack Overflow. Among the 16 participants who had tried forums, only 3 participants contributed to it (e.g., posting a thread or replying on others' threads).

**Online coding tutorials:** Several participants mentioned that they attempted to self-teach programming by following online coding tutorials. Among these tutorials, step-by-step YouTube videos appeared to be the most popular among our participants (10/23), followed by text-based interactive tutorials (8/23) that included *Codecademy, FreeCodeCamp,* and *CSS tricks.* Participants mentioned trying out online tutorials particularly for web development topics.

**Popular press:** Lastly, several participants (9/23) mentioned that they subscribed to technology-related online content to broaden their perspective of cutting edge technology and developments. These resources included technology-related podcasts and popular press, such as New *Scientist Magazine, Peter Diamandis's blog, Tech Insider, Forbes, Bloomberg, CNN, Guardian, TechCrunch*, and company newsletters.

## PERCEPTIONS OF THE LEARNING PROCESS AND FEELINGS OF FAILURE

As described above, our participants had engaged in a variety of informal and formal learning strategies based on recommendations from developers or other technical experts or by searching online. In reflecting back on their original motivations to mainly improve technical conversations, unfortunately, most participants felt that they did not get much benefit from investing the time and effort on these programming resources and expressed feelings of failure. In fact, only 6 participants reported that learning programming was useful for technical conversations, and only 3 participants felt confident enough to mention programming as a skill on their CV or during a job interview.

In this section, we present a synthesis of the six common reasons that conversational programmers felt they failed when using modern learning resources (summarized in Table 3).

### Takes too much time

Since conversational programmers were not required to write code as their regular day job, the time they could commit to learn programming was limited (consistent with other studies on adult learners [24,59]). Whether or not using a certain resource would be time-consuming was a concern raised by most of the participants.

Although formal approaches provided a systematic learning environment with assistance from an instructor, our participants did not consider them to be practical because they required the most time commitment. For example, most participants (21/23) did not sign up for in-person courses because they felt it was not necessary to take a course or they simply

did not have enough time to take it. Even though some participants did sign up for MOOCs and other online courses (6/23) and could leverage the convenience of distance learning, most participants ended up being busy with their day job and found it difficult to maintain focus and commit time for completion:

*I am learning JavaScript in CS50. It's a real Harvard lecture, so you have students from Harvard attending it and they just film the thing. But I have given up on it several times... This is my fourth time taking CS50, or fourth time attempting to... Every time I get caught up with other work or I'm too busy. (P7)*

Although informal resources were perceived to be easier to use, they could also be time-consuming because conversational programmers did not have enough background to *"have the vocabulary to phrase the questions" (P18)*. They often ended up spending hours and *"finding nothing that's really useful" (P6)*. For example, one participant complained that going through non-relevant YouTube videos could be a huge time sink:

*So sometimes there might be stuff [in videos] you already know or stuff that you just do not care about. Sometimes it could even be an advertisement. A lot of garbage, no kidding. But you only know it after watching [the whole video]. (P14)*

**Too much focus on syntax and logic**

In their initial learning approach, conversational programmers were influenced by many preconceptions such as, *"to learn programming, you have to write code. It's just like learning to drive a car, you cannot learn without running a car" (P18),* or they feel like they *"have to start from the beginning" (P8)*. Therefore, the majority of participants (18/23) had devoted some time to learn to code in a specific language.

However, after signing up for an online course or using online tutorials to learn a specific programming language, not many participants found it helpful enough with building common ground in technical conversations. For example, P11 admitted that going through the online coding tutorials did not help so much with understanding the big picture:

*I think they [coding tutorials] were very good like instructionally… But, what I definitely needed is to be able to talk…just being able to write code, I find that I am missing out on some kind of larger understanding. (P11)*

Another participant who paid time and money to attend an introductory level bootcamp mentioned that she *"wouldn't take it again"* because she felt that these bootcamps were designed for people pursuing careers as software developers and often became more technical than she expected:

*It [the bootcamp] was overwhelming…the coding skills they taught is to enable somebody to parachute into a web development job...not what I expected...(P6)*

One of our participants who was a university administrative staff and worked closely with students in CS, described her experience after attending a coding workshop in Python:

*I did the "Python Ladies Learning Code", an all-day introductory workshop…I thought it was obviously super helpful for me to understand a little bit about programming since I'm talking to CS major students all the time... But I don't know if it actually helps. I mean it's so basic level coding, right? Although I had several lines of codes working and printed sentences on the screen in that workshop, I can't recall anything tangible now. (P15)*

**Explanations are not relevant**

Several participants mentioned that when they were interacting with programming-related resources, their main goal was to seek conceptual and application-related explanations:

*... when I am learning about cache and cookies [on online documentation], I don't want to know if I have to use 'loop' or 'if-else' or anything like that, I want to know what it can do for me, like the user side of it. (P9)*

Participants gave up on resources that did not give enough information on the bigger picture of concepts:

*I have given up on a YouTube channel because they were deviating from what I want to learn and they were getting like a lot deeper than I wanted. And especially that channel was like for people who want to do the programming…they spent less time for the bigger concept. (P5)*

Understanding the limitations and benefits of programming or technology choices was important for conversational programmers, but such explanations were not always available in programming learning resources:

*...if they [developers] are saying, "Oh, we are going to use a library X to do this", I think it would be good to know, ok…what does that mean, how much time and money does it take to use library X, how much does it improve performance of the database? I searched [for] any websites that have the information out there, and haven't really seen anything related to that. (P20)*

In addition to the limitations and benefits, participants mentioned that they also needed to know the difference between certain terms or to connect the terms to a working process:

*Sometimes I need to know like how it's different from something else or how it relates to something else. For example, like machine learning and deep learning.... I saw a blog on that, talking about...like neural networks... I can't remember, but like very technical and low-level explanations. (P15)*

Lastly, participants also sought explanations on software engineering processes and development structures. For example, one participant who was an HR coordinator explained how she wanted to know about *"how development teams are structured"* since she was *"in charge of hiring and interviewing future developers to the company." (P14)*.

Since the target users of introductory learning resources are traditional programmers who will build artifacts [26,31,54], most of these resources concentrate on teaching syntax and logic, and problem solving skills. As a result, conversational programmers in our study struggled to find relevant conceptual and application-related explanations in these resources.

**Difficult to assess the content's reliability**

Professional programmers or end-user programmers who write code can often use "trial and error" to verify whether a tip or suggestion from a learning resource actually works in

code [3,15,30]. However, conversational programmers explained that they did not have the opportunity use "trial and error" in conversations and the stakes were higher in getting accurate definitions and explanations from a resource.

Although online search was popular among conversational programmers, they did not often trust the search results and still wanted confirmation from colleagues or friends:

*There is so much garbage on the internet that if you search something that does not look like an incredible website then I want to verify it with a human being. And all my colleagues would just be like, "Hey, stop googling it!" (P1)*

Participants also doubted the credibility of community-based sites. For example, only half of the participants who tried forums (8/16) felt that they got anything useful from forums—the rest had strong negative opinions:

*…when I browse the questions [on forums], the people who originally posted do not give follow-up details on whether the answers worked or not…I understand part of it and then I am not sure if the person actually got it [to work]…(P9)*

In addition, participants raised concerns about whether or not to trust the accuracy of the content being presented in other resources, such as YouTube videos. One participant who was a marketing coordinator expressed doubts on the utility of watching free videos and stated a preference for instead relying on paid courses on sites like Lynda.com:

*It's hard to gauge if these people [video authors] are professionals or if this is an accurate way of doing it. So I use Lynda.com now, our company has a subscription for that and lots of my colleagues are using it. (P8)*

**Feelings of social isolation**

Since most of our participants were domain experts in a non-technical role, they tended to stay away from certain resources because they felt uncomfortable, stressful, and isolated in environments where the target learners were perceived to be more experienced or even professional programmers.

One of the participants who attended a bootcamp found it stressful to keep up with people who already had some knowledge of programming:

*Because my classmates were not newbies at what they were learning...the level that I had to try to reach to them [was hard] … I was constantly trying to catch up and understand. (P6)*

Despite the convenience of relying on experts, some participants described the social cost of bothering people who were already overworked by asking them naive questions. For example, one participant who was learning through Codecademy said that he would never ask any of his developer colleagues for help:

*I mean, I know any one of my colleagues could solve any of my problems, in about six seconds. But the point is not to ... They already have their own work to do and for me, this is again, it's not critical to what I do, and it's not worth spending the company resources to do that. And again, my friends know I don't code, so they don't want to help me with that. (P17)*

Sometimes when conversational programmers referred to an expert for help, they were hesitant to ask follow-up questions because they *"did not want to look stupid" (P8)*. One participant even said that, *"I pretend I kind of understand what he [the expert] is talking about and rather figure it out later by myself" (P4)*. It could also be embarrassing to ask an expert to re-explain a concept he or she had previously described:

*What I hate is like they explain it to me and I still don't get it. That's the worst. Because with the internet, it doesn't matter. I can keep googling. With people, it's just, I don't know, it's a little embarrassing. (P15)*

When using online learning resources and forums where there was less of a direct social cost, participants reported that sometimes they still felt like an outsider. None of the participants had contributed to developers' communities like Stack Overflow. Their general perceptions were negative:

*[Stack Overflow] They're often populated by developers, not for the lay person. So again, the assumption that you understand concepts and things already to a certain level is already inherent in there. And quite frankly, a lot of developers are jerks. It can be pretty toxic. Some people are even like "Okay, this is not the place you should ask". (P13)*

**Easy to forget details without a direct application**

Lastly, participants had feelings of failure when trying to learn programming as they tended to *forget* what they learned over time.

For example, one participant who tried Codecademy to learn JavaScript said he would not do it again because he kept forgetting the concepts without applying the knowledge:

*Programmers learn and write code on a regular basis. But if you don't use it, you just forget it. So why would I put the effort to learn something that would then just get incredibly rusty and then forget half of it in six months anyway? (P17)*

Similarly, another participant who took an introductory course to learn *"fundamentals of HTML"* on Lynda.com said that it was easy for him to forget the concepts because he skipped the coding exercises for the sake of time:

*They [Lynda.com] have optional exercises after each lecture... But I mean all I want is just some conceptual level understanding of what's going on. So I skipped the exercise. Sometimes you are just like "It looks easy. I'll just test it later" and then you never do. It turns out that I just forget the concepts very quick. (P8)*

In some cases, conversational programmers could retain what they learned for a short-term project or to satisfy an immediate need, but not beyond. For example, an entrepreneur who once hired developers to build a website for her company explained this phenomenon:

*I only learn it when I need to use it. And then I promptly forget it all. When we built our company's first website, I spent like 3 days locked in my room to learn some basic stuff like WordPress, HTML. But I can't recall anything now at all because I didn't use it for a long time. (P1)*

Sometimes participants learned terminologies in technical conversations but would forget them after the first exposure. For example, one participant explained how he had to:

*...look up the term again a month later because I just skimmed the first paragraph to get a general idea [the first time] ...but, I forgot a lot afterwards...(P13)*

In addition, one participant even felt nervous when she tried to recall the definition of a *"database"*, which she had learned recently from a coding bootcamp:

*My palms are sweating...I am just nervous because I learned [about databases] two weeks ago and I cannot remember much right now. I might have to sign up for the same course again. (P6)*

As shown above, there were six key reasons why conversational programmers developed feelings of failure in their pursuit of learning programming (summarized in Table 3).

## THE PARADOX OF LEARNING PROGRAMMING

In the previous section, we examined how conversational programmers approached learning programming and how most of them felt like they failed, even after investing a lot of time and effort. However, our findings reveal an interesting paradox in the participants' perceptions of programming: despite feelings of failure in their attempts to improve technical conversations, the majority (19/23) still wanted to *keep learning programming* in the future if appropriate learning resources were available. For example, a product manager described this as, *"a short path with acceptable opportunity cost" (P13)*. Another participant reported that she only wanted to learn what is related with her project in the future:

*I will definitely keep learning [programming] in the future, because then you have a better understanding of the terminology that's being used, and it saves much work for your job. But I don't want to start everything from scratch, it's like a deep pool. I only want to learn what's related with my project. (P19)*

A common reason identified by the participants was that having some background in programming allowed them to better understand the work of their technical team members and build empathy for them:

*[programming] doesn't help so much with the technical conversation... But I do have the feeling now that their [developers'] work is extremely hard after I learned. I think it's given me a lot more empathy on understanding that it's not easy to do what you want just because you envision being able to do it. (P7)*

Another advantage of learning programming was having a better sense of being able to estimate implementation time:

*I feel like I'm much more generous in terms of time now. I understand it might take forever to write the small change. It's a struggle to write even a little bit of code. It's all about debugging and unknown errors. (P8)*

Moreover, participants felt that they earned more respect from developers as well. Learning programming helped them gain credibility and build rapport with developers:

*The programming people tend to be not interested in talking to me [before]…Being a coder is a badge of honor, people respect me more [now]. (P3)*

Although the majority of participants failed in learning programming, a small number of them did achieve success using resources where they could connect with other conversational programmers. For example, a participant who was a visual designer actively searched and reached out to other designers who were learning programming: *"I'm on a Slack group, and all of th ese Facebook groups and LinkedIn groups". (P2)*

| |
|---|
| **Takes too much time:** Investing in learning programming ended up requiring more time than what participants wanted to devote given their busy schedules. |
| **Too much focus on syntax and logic:** Most of the resources focused on programming syntax and logic which did not directly help participants with their technical conversations. |
| **Explanations are not relevant:** The conceptual and application-related explanations desired by the participants were not always relevant nor available in the learning resources. |
| **Difficult to assess the content's reliability:** Participants did not feel confident enough to assess whether a given resource contained accurate and reliable content. |
| **Feelings of social isolation:** Resources and learning environments that target CS students or professional programmers often created feelings of social isolation among participants. |
| **Easy to forget details:** It was easy for participants to forget programming definitions and details because they did not apply what they learned directly on-the-job. |

**Table 3. Six common reasons for feelings of failure among conversational programmers when using modern resources**

Another participant who worked as a library archivist and collaborated with developers on a project to digitize materials explained how she benefited by being in the same room as other archivists and librarians learning programming:

*I think we often don't receive enough training…and so those sorts of [technical] workshops are great. It is a nice opportunity to work through problems with other people who also need this skill and don't have the background in it. It's nice to have someone in a similar situation as me to talk to. (P11)*

In summary, our findings reveal a paradox in conversational programmers' perceptions of programming in that while they feel like they failed, they still acknowledged the value of learning programming under certain circumstances.

## DISCUSSION

Our findings overall illustrate that the learning needs and constraints of conversational programmers had some similarities to other adult learners who have rigid schedules [6,24,59] and prefer informal learning approaches [14,41]. However, we also found some critical differences among these groups of learners. For example, in contrast to end-user programmers who may prefer resources with rich implementation details and "ready-to-go" examples [15], conversational programmers found such details to be distracting and preferred to see more conceptual explanations. Although CS teachers also do not need to build artifacts [43], they differ from conversational programmers as their needs are still more syntax-oriented—they need to be able to teach low-level concepts and create and grade coding assignments.

In this paper, our main contribution has been in providing novel insights into how a broad range of professionals who

do not need to write code (e.g., archivist, artist, entrepreneur, psychologist, event manager, medical instructor and visual designer) use formal and informal approaches to learn programming. We have also contributed insights into reasons why modern learning resources fail these conversational programmers in their pursuits to improve technical conversations. We now reflect on the mismatch of expectations that conversational programmers experience and how HCI and learner-centered design [24] approaches can play a pivotal role in better supporting this emerging learner population.

### A Mismatch of Expectations

We learned that although almost all of the conversational programmers in our study were interested in learning programming to improve their conversations, in the end, about 75% of the participants did not feel that they achieved this goal. Their narratives illustrated a mismatch of expectations that manifested in two ways, described below.

#### *Is programming knowledge even necessary?*

The first mismatch occurred because conversational programmers often assumed that learning programming would help them with grounding in technical conversations. Our participants described their attempts in collectively learning over 20 different programming languages even though they did not need to write any code. However, their descriptions and challenges of technical conversations revealed that these learners were more interested in establishing a conceptual understanding of terminologies, benefits and limitations of technologies, and tradeoffs in software design and implementation decisions. Therefore, is pursuing programming even the right approach for conversational programmers?

Future work could investigate why such misconceptions form about programming in the first place. Perhaps with all of the recent excitement around *programming for all* or *computational thinking* being popularized in the press [48], people tend to associate anything technical with programming [16]. Another possibility is that people assume that just because they are talking to programmers, they need to understand the "programmers' language". But, the kinds of expertise and vocabulary that developers possess can take years of education or experience to develop, so it is not realistic to expect newcomers to master all the concepts with introductory learning resources.

On the other hand, if conversational programmers do not learn programming at all, is it even possible for them to understand technical decisions, tradeoffs, or higher-level concepts, such as machine learning or cloud-based architecture? It may be the case that learning the basics of programming and some technical jargon are important dimensions of establishing this common ground that conversational programmers seek to establish [9,60].

#### *Is my chosen learning resource even appropriate?*

The second mismatch ensued when conversational programmers interacted with the same modern resources that are typically used by learners who want to eventually build artifacts or solve computational problems. Such resources often follow a more structured syntax-oriented curriculum (known as "programming-first approach") of introductory computer science programs in universities [61]. All of this investment in learning programming through these resources created a *rabbit hole* effect for conversational programmers as they were led down a path of struggling with programming syntax and all of the other issues that novice programmers encounter [34] while not getting much direct benefit for improving their technical conversations.

Still, despite the mismatch in expectations and feelings of failure, the majority of conversational programmers wanted to keep learning programming if appropriate learning resources were available, which suggests that HCI can play a key role in designing suitable learner-centered resources.

### Design Opportunities for Supporting Conversational Programmers

Here we consider the design implications of our findings and how we can better support conversational programmers.

#### *Facilitating Discovery of Relevant and Reliable Content*

Given the challenges that conversational programmers face in spending time on learning resources and in sifting through irrelevant and unreliable search results, future research can look into facilitating discovery of relevant and reliable content. For example, we can explore how to create Wikipedia-like curated overviews with small examples that are focused on specific application areas. The goal here should be to make them easily "skimmable" in a few minutes—similar approaches have recently been seen in resources such as *wikiHow* [62] that focus on small bite-sized tutorials. How can we create a wikiHow-like site for facilitating discovery of programming concepts, and how would this scale?

At the same time, authoritativeness of learning resources is important for this learner population and "trial and error" [2,13,15,28] approaches that work for novice or end-user programmers do not work for conversational programmers. These learners may find little success in searching for programming and debugging help in ad-hoc blogs and forums where they can plug-and-play solutions. Instead, conversational programmers can benefit from resources and explanations that are endorsed by leaders in the field to have confidence that they are high-quality materials. There are opportunities for future work to investigate who these leaders would be and how would they make contributions towards endorsing a particular resource.

#### *Explaining Concepts without Syntax and Logic*

A key challenge that our findings raise for the HCI community to consider is, can we actually teach someone useful programming concepts *without* focusing on syntax and logic? What would that even mean? What would be the advantages or disadvantages of doing so?

A popular approach that has been explored in research and practice is the design of novice-friendly "drag-and-drop" [42] programming languages and systems such as *Alice* [11], *Scratch* [45], and *Code.org* [63] to make programming more attractive for children [39,57] and other novices [21]. How-

ever, none of our participants were familiar with such environments and would likely not find them useful for improving their technical conversations because these approaches still largely focus on the mechanics of programming.

Another approach may be to design courses with emphasis on more conceptual instruction of computing concepts without writing code [2,20,35,40,56]. For example, Cornell University has recently experimented with a non-programming introduction to CS via concepts, such as in NLP and AI [35]. It may be possible to extend such an approach outside of the classroom to also teach conversational programmers useful concepts without getting into the mechanics of syntax. Another useful augmentation here may be to teach conversational programmers how to *talk* about a particular concept in the context of a real-world development scenario. For example, some online dictionaries offer the ability to not just view the definition of a word, but to see how the word may be used in a sentence. It may be fruitful to explore how such interactive reference resources could be created for connecting real-world context with programming-related concepts for conversational programmers.

*Generating Executive Summaries and Visual Explorations*

Given that conversational programmers may only have an ephemeral need to understand and apply some concepts, future research can explore how to design interactive high-level executive summaries or allow for more visual explorations of such concepts. One approach could be presenting a comparative or competitive analysis like an executive report containing the pros and cons to be delivered to a business executive to help them make decisions. For instance, such a summary could make it easy to glance at the pros and cons of neural networks or weigh the benefits of using Amazon's vs. Google's cloud services.

At the code level, perhaps there is a need for more visual explorations like interactive neural net explorations [5], *explorable explanations* [64] or algorithm animation [4] to give learners interactive visual ways to learn to gain intuitions without writing any code, which is similar to the idea of data analysis tools or prototyping tools that allow people to explore ideas and possibilities without writing code [65,66].

*Building Conversational Programmers' Own Communities*

We found that conversational programmers expressed feelings of isolation when trying to learn from resources designed for professional or end-user programmers. As discussed above, there is some indication that the recommendations on learning resources from other programmers create a mismatch of expectations. Therefore, it would be worth exploring if the perceptions of conversational programmers would be different if the recommendations came from other conversational programmers similar to them. There is an opportunity here for HCI/CSCW to explore the benefits and drawback of social and personalized recommendations for this learner group.

One design opportunity may be in creating a welcoming community of like-minded peers and mentors, who are perhaps not the stereotypical computer "geeks" or "insiders" as described by many of our participants. There already are learning communities emerging for certain non-traditional learners, such as scientists [58], CS teachers [44], and even product managers [9]. Similarly, we could build conversational programmers' own communities through formal workshops (e.g., dedicated bootcamps) or through online resources and meetups. Learners can receive suggestions and mentorship from experienced conversational programmers who have gone through the same process or are currently going through it. These communities can perhaps evaluate existing resources from the perspective of their domain (e.g., similar work has been done to evaluate programming systems using techniques such as heuristic evaluation [33] ).

### Limitations and Future Work

Our focus was only on perceptions and learning strategies; future work can use controlled studies to formally explore learning outcomes of different interventions and approaches. Although we had a diverse set of participants in terms of job roles and experiences, we did not explore gender, occupation-specific learning goals, or other demographic differences in responses. In addition, since our recruitment criteria explicitly mentioned an attempt to learn programming, we did not have the chance to investigate "conversational technical non-programmers", who did technical communication with programmers but never attempted to learn programming. This population is worthwhile to study in the future.

More importantly, when we talk about "grounding in communication" [10], there are actors on both sides (technical and non-technical) and our results so far paint a picture from only one side. It should not be solely the job of conversational programmers to make an investment in extra on-the-job learning. Great software engineers should be both productive at the job and good at communicating [37,50]. Moreover, they should not only focus on effectively working with other technical people, but also on better explaining their decisions to people who are non-engineers. Our study opens a path for future research to bridge the gap in technical conversations from developers' perspectives as well.

## CONCLUSION

In conclusion, we have contributed insights from conversational programmers across a wide range of job roles who experience challenges and try to learn programming to improve their conversations. In particular, we have described their learning approaches and struggles and highlighted six reasons why modern resources designed for traditional learners, such as CS students and professional programmers, are not appropriate for this learner population. We have also highlighted ways in which HCI can play a pivotal role in designing learning resources and interactions that are suitable not only for conversational programmers but also other members of society who are increasingly wanting to develop programming and technical literacy.

## ACKNOWLEDGMENTS

This research was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC). We thank Prashant Shashikumar and Azadeh Zamani Esfahani.

## REFERENCES


1. Khaled Albusays and Stephanie Ludi. 2016. Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE '16), 82–85. https://doi.org/10.1145/2897586.2897616
2. Tim Bell, Jason Alexander, Isaac Freeman, and Mick Grimley. 2009. Computer science unplugged:school students doing real computing without computers. *New Zealand Journal of Applied Computing and Information Technology* 13, 1: 20–29.
3. Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '09), 1589–1598. https://doi.org/10.1145/1518701.1518944
4. Marc H. Brown. 1988. *Algorithm Animation*. MIT Press, Cambridge, MA, USA.
5. Shan Carter and Daniel Smilkov. Tensorflow — Neural Network Playground. Retrieved January 5, 2018 from http://playground.tensorflow.org
6. Polina Charters, Michael J. Lee, Andrew J. Ko, and Dastyni Loksa. 2014. Challenging Stereotypes and Changing Attitudes: The Effect of a Brief Programming Encounter on Adults' Attitudes Toward Programming. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 653–658. https://doi.org/10.1145/2538862.2538938
7. Parmit K. Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J. Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 251–259. https://doi.org/10.1109/VLHCC.2015.7357224
8. Parmit K. Chilana, Rishabh Singh, and Philip J. Guo. 2016. Understanding Conversational Programmers: A Perspective from the Software Industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16), 1462–1472. https://doi.org/10.1145/2858036.2858323
9. Ellen Chisa. 2014. Evolution of the Product Manager. *Queue* 12, 9: 40:40–40:47. https://doi.org/10.1145/2674600.2683579
10. Herbert H. Clark and Susan E. Brennan. 1991. Grounding in Communication. In *Perspectives on Socially Shared Cognition*, Lauren Resnick, Levine B, M. John, Stephanie Teasley and D. (eds.). American Psychological Association, 13–1991.
11. Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. In *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges* (CCSC '00), 107–116.
12. Juliet Corbin and Anselm Strauss. 2014. *Basics of Qualitative Research*. SAGE.
13. Sarah D'Angelo and Andrew Begel. 2017. Improving Communication Between Pair Programmers Using Shared Gaze Awareness. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 6245–6290. https://doi.org/10.1145/3025453.3025573
14. Brian Dorn and Mark Guzdial. 2006. Graphic Designers Who Program as Informal Computer Science Learners. In *Proceedings of the Second International Workshop on Computing Education Research* (ICER '06), 127–134. https://doi.org/10.1145/1151588.1151608
15. Brian Dorn and Mark Guzdial. 2010. Learning on the Job: Characterizing the Programming Knowledge and Learning Strategies of Web Designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10), 703–712. https://doi.org/10.1145/1753326.1753430
16. Chase Felker, Meg Charlton, and Joshua Oliver. 2013. Maybe Not Everybody Should Learn to Code. *Slate*. Retrieved January 5, 2018 from http://www.slate.com/articles/technology/future_tense/2018/01/there_is_no_such_thing_as_the_blockchain.html
17. Sally Fincher. 2015. What Are We Doing when We Teach Computing in Schools? *Commun. ACM* 58, 5: 24–26. https://doi.org/10.1145/2742693
18. J. Michael Fitzpatrick, Ákos Lédeczi, Gayathri Narasimham, Lee Lafferty, Réal Labrie, Paul T. Mielke, Aatish Kumar, and Katherine A. Brady. 2017. Lessons Learned in the Design and Delivery of an Introductory Programming MOOC. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '17), 219–224. https://doi.org/10.1145/3017680.3017730
19. Andrea Forte and Mark Guzdial. 2005. Motivation and Non-Majors in Computer Science: Identifying Discrete Audiences for Introductory Courses. *IEEE Transactions on Education* 48, 2: 248–253. https://doi.org/10.1109/TE.2004.842924
20. Kenneth J. Goldman. 2004. A Concepts-first Introduction to Computer Science. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '04), 432–436. https://doi.org/10.1145/971300.971446
21. Paul Gross and Kris Powers. 2005. Evaluating Assessments of Novice Programming Environments. In *Proceedings of the First International Workshop on Computing Education Research* (ICER '05), 99–110. https://doi.org/10.1145/1089786.1089796
22. Philip J. Guo. 2017. Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 7070–7083. https://doi.org/10.1145/3025453.3025945

23. Mark Guzdial. 2003. A Media Computation Course for Non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE '03), 104–108. https://doi.org/10.1145/961511.961542
24. Mark Guzdial. 2015. Learner-Centered Design of Computing Education: Research on Computing for Everyone. *Synthesis Lectures on Human-Centered Informatics* 8, 6: 1–165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033
25. Mark Guzdial and Andrea Forte. 2005. Design Process for a Non-majors Computing Course. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '05), 361–365. https://doi.org/10.1145/1047344.1047468
26. Carolin D. Hardin and Matthew Berland. 2016. Learning to Program Using Online Forums: A Comparison of Links Posted on Reddit and Stack Overflow (Abstract Only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (SIGCSE '16), 723–723. https://doi.org/10.1145/2839509.2851051
27. Kyle J. Harms, Evan Balzuweit, Jason Chen, and Caitlin Kelleher. 2016. Learning Programming from Tutorials and Code Puzzles: Children's Perceptions of Value. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 59–67. https://doi.org/10.1109/VLHCC.2016.7739665
28. Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 2016. A Study of Code Design Skills in Novice Programmers Using the SOLO Taxonomy. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (ICER '16), 251–259. https://doi.org/10.1145/2960310.2960324
29. Geoffrey James. 2017. Why Coding Bootcamps Don't Work. *Inc.com*. Retrieved January 5, 2018 from https://www.inc.com/geoffrey-james/why-coding-bootcamps-dont-work.html
30. Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 1265–1276. https://doi.org/10.1145/3025453.3025626
31. Ada S. Kim and Andrew J. Ko. 2017. A Pedagogical Analysis of Online Coding Tutorials. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '17), 321–326. https://doi.org/10.1145/3017680.3017728
32. Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* 43, 3: 21:1–21:44. https://doi.org/10.1145/1922649.1922658
33. Michael Kölling and Fraser McKay. 2016. Heuristic Evaluation for Novice Programming Systems. *Trans. Comput. Educ.* 16, 3: 12:1–12:30. https://doi.org/10.1145/2872521
34. Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (ITiCSE '05), 14–18. https://doi.org/10.1145/1067445.1067453
35. Lillian Lee. 2002. A Non-Programming Introduction to Computer Science via NLP, IR, and AI. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1* (ETMTNLP '02), 33–38. https://doi.org/10.3115/1118108.1118113
36. Michael J. Lee and Andrew J. Ko. 2015. Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (ICER '15), 237–246. https://doi.org/10.1145/2787622.2787709
37. Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. 2015. What Makes a Great Software Engineer? In *Proceedings of the 37th International Conference on Software Engineering - Volume 1* (ICSE '15), 700–710.
38. Yihan Lu, I-Han Hsiao, and Qi Li. 2016. Exploring Online Programming-Related Information Seeking Behaviors via Discussion Forums. In *2016 IEEE 16th International Conference on Advanced Learning Technologies (ICALT)*, 283–287. https://doi.org/10.1109/ICALT.2016.63
39. John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by Choice: Urban Youth Learning Programming with Scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '08), 367–371. https://doi.org/10.1145/1352135.1352260
40. Joe Marks, William Freeman, and Henry Leitner. 2001. Teaching Applied Computing Without Programming: A Case-based Introductory Course for General Education. In *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '01), 80–84. https://doi.org/10.1145/364447.364547
41. Victoria J. Marsick and Karen E. Watkins. 2001. Informal and Incidental Learning. *New Directions for Adult and Continuing Education* 2001, 89: 25–34. https://doi.org/10.1002/ace.5
42. Paul Medlock-Walton, Kyle J. Harms, Eileen T. Kraemer, Karen Brennan, and Daniel Wendel. 2014. Blocks-based Programming Languages: Simplifying Programming for Different Audiences with Different Goals. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 545–546. https://doi.org/10.1145/2538862.2538873
43. Lijun Ni and Mark Guzdial. 2012. Who AM I?: Understanding High School Computer Science Teachers' Professional Identity. In *Proceedings of the 43rd ACM*

*Technical Symposium on Computer Science Education* (SIGCSE '12), 499–504. https://doi.org/10.1145/2157136.2157283
44. Lijun Ni, Mark Guzdial, Allison Elliott Tew, Briana Morrison, and Ria Galanos. 2011. Building a Community to Support HS CS Teachers: The Disciplinary Commons for Computing Educators. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education* (SIGCSE '11), 553–558. https://doi.org/10.1145/1953163.1953319
45. Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11: 60–67. https://doi.org/10.1145/1592761.1592779
46. Christopher Scaffidi, Aniket Dahotre, and Yan Zhang. 2012. How Well Do Online Forums Facilitate Discussion and Collaboration Among Novice Animation Programmers? In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (SIGCSE '12), 191–196. https://doi.org/10.1145/2157136.2157195
47. Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the Numbers of End Users and End User Programmers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, 207–214. https://doi.org/10.1109/VLHCC.2005.34
48. Esther Shein. 2014. Should Everybody Learn to Code? *Commun. ACM* 57, 2: 16–18. https://doi.org/10.1145/2557447
49. Jonathan Sillito, Frank Maurer, Seyed Mehdi Nasehi, and Chris Burns. 2012. What Makes a Good Code Example?: A Study of Programming Q&A in StackOverflow. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)* (ICSM '12), 25–34. https://doi.org/10.1109/ICSM.2012.6405249
50. Edward K. Smith, Christian Bird, and Thomas Zimmermann. 2016. Beliefs, Practices, and Personalities of Software Engineers: A Survey in a Large Software Company. In *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE '16), 15–18. https://doi.org/10.1145/2897586.2897596
51. Elliot Soloway, Mark Guzdial, and Kenneth E. Hay. 1994. Learner-centered Design: The Challenge for HCI in the 21st Century. *interactions* 1, 2: 36–48. https://doi.org/10.1145/174809.174813
52. Sabine Sonnentag, Cornelia Niessen, and Sandra Ohly. 2004. Learning at work: training and development. *International review of industrial and organizational psychology* 19: 249–290.
53. Phit-Huan Tan, Choo-Yee Ting, and Siew-Woei Ling. 2009. Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. In *Proceedings of the 2009 International Conference on Computer Technology and Development - Volume 01* (ICCTD '09), 42–46. https://doi.org/10.1109/ICCTD.2009.188
54. Kyle Thayer and Andrew J. Ko. 2017. Barriers Faced by Coding Bootcamp Students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (ICER '17), 245–253. https://doi.org/10.1145/3105726.3106176
55. Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How Do Programmers Ask and Answer Questions on the Web? (NIER Track). In *2011 33rd International Conference on Software Engineering (ICSE)*, 804–807. https://doi.org/10.1145/1985793.1985907
56. Mark Urban-Lurain and Donald J. Weinshank. 2000. Is there a role for programming in non-major computer science courses? In *30th Annual Frontiers in Education Conference. Building on A Century of Progress in Engineering Education. Conference Proceedings (IEEE Cat. No.00CH37135)*, T2B/7-T2B11 vol.1. https://doi.org/10.1109/FIE.2000.897590
57. Linda Werner, Shannon Campe, and Jill Denner. 2012. Children learning computer science concepts via Alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 427–432.
58. Greg Wilson. 2006. Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive. *Computing in Science Engineering* 8, 6: 66–69. https://doi.org/10.1109/MCSE.2006.122
59. Chi Zhang and Guangzhi Zheng. 2013. Supporting Adult Learning: Enablers, Barriers, and Services. In *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education* (SIGITE '13), 151–152. https://doi.org/10.1145/2512276.2512323
60. Will Non-Technical Product Managers Become Obsolete? Retrieved January 5, 2018 from https://www.forbes.com/sites/quora/2017/01/03/will-non-technical-product-managers-become-obsolete
61. ACM Curricula Recommendations. Retrieved January 5, 2018 from http://www.acm.org/education/curricula-recommendations
62. wikiHow - How to do anything. Retrieved January 5, 2018 from http://www.wikihow.com/Main-Page
63. Anybody can learn | Code.org. Retrieved January 5, 2018 from https://code.org/
64. Explorable Explanations. Retrieved January 5, 2018 from http://explorabl.es/
65. Business Intelligence and Analytics | Tableau Software. Retrieved January 5, 2018 from https://www.tableau.com/
66. Prototypes, Specifications, and Diagrams in One Tool | Axure Software. Retrieved January 5, 2018 from https://www.axure.com/