Investigating Learning Strategies of Conversational Programmers

by

April Yi Wang

B.Eng., Zhejiang University, 2016

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

in the

School of Computing Science
Faculty of Applied Science

© April Yi Wang 2018
SIMON FRASER UNIVERSITY
Summer 2018
Approval

Name: April Wang
Degree: Master of Science (Computing Science)
Title: Understanding and Lowering the Learning Barriers for Conversational Programmers

Examining Committee:
Chair: Greg Baker
Senior Lecturer

Parmit Chilana
Senior Supervisor
Professor

Lyn Bartram
Supervisor
Associate Professor

Philip Guo
External Examiner
Assistant Professor
Department of Cognitive Science
University of California San Diego

Date Defended: July 24 2018
Ethics Statement

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

a. human research ethics approval from the Simon Fraser University Office of Research Ethics

or

b. advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University

or has conducted the research

c. as a co-investigator, collaborator, or research assistant in a research project approved in advance.

A copy of the approval letter has been filed with the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Simon Fraser University Library
Burnaby, British Columbia, Canada

Update Spring 2016
Abstract

Conversational programmers represent a class of learners who come from non-CS backgrounds and try to learn programming mainly to improve their participation in technical conversations. We carried out interviews with 23 conversational programmers (e.g., advertising manager, psychologist, library archivist) to better understand the challenges they face in technical conversations and their perceptions of learning programming. Among our key findings, we found that learning programming and consulting modern learning resources that focus on programming syntax and logic was not helpful for these learners as their end-goal was not to build artifacts and needed to understand concepts well enough to participate in technical conversations. Based on our findings, we used a user-centered design process to develop JargonAid, a community-curated online dictionary that allows authors to add simple explanations and example conversations for technical concepts. For our evaluation, we compared community-curated explanations in JargonAid with formal explanations (e.g., from textbooks, online documentation). We found that conversational programmers perceived community-curated explanations to be easier to understand, allowed them to maintain focus, and made them more confident about participating in follow-up conversations. We discuss the potential of using community-curated explanations as a learning tool for conversational programmers and reflect on the advantages and disadvantages of our approach.

Keywords: learner-centered design; conversational programmers; learning programming;
Dedication

To my parents, Haihong Cheng and Haijun Wang
Acknowledgements

Above all, I would like to express my deep gratitude to my supervisor, Dr. Parmit Chilana for her patient guidance, enthusiastic encouragement, without which none of this thesis would have ever happened.

I would also like to thank my committee members, Dr. Philip Guo and Dr. Lyn Bartram, for their valuable and constructive insights that helped to refine this thesis. Laton Vermette, Ryan Mitts, Megha Nawhal, Kimia Kiani, Rimika Chaudhury, Mahya Sadeghi and other fellow lab members at SFU HCI Lab: thank you for providing advice and support. And to all of the numerous people who assisted me and participated in my research.

Finally, I wish to thank my parents Haihong Cheng, Haijun Wang for their unconditional love, support and patience throughout my education.
Table of Contents

Approval .......................................................................................................................... ii
Ethics Statement ........................................................................................................... iii
Abstract ........................................................................................................................... iv
Dedication ........................................................................................................................ v
Acknowledgements ......................................................................................................... vi
Table of Contents .......................................................................................................... vii
List of Tables .................................................................................................................. ix
List of Figures ............................................................................................................... x

Chapter 1. Introduction ....................................................................................................... 1

Chapter 2. Related Work .................................................................................................. 4
2.1. Studies of Non-traditional Programmers ................................................................. 4
2.2. Formal Learning Environments for Programming .................................................... 5
2.3. Informal Learning Resources for Programming ....................................................... 6
2.4. Professionally-oriented Online Communities ........................................................... 6

Chapter 3. Understanding Learning Strategies of Conversational Programmers. 8
3.1. Research Approach and Method ............................................................................... 8
3.1.1. Participants and Recruitment ............................................................................... 8
3.1.2. The Interview Instrument .................................................................................... 9
3.1.3. Data Analysis ....................................................................................................... 10
3.1.4. Presentation of Results ......................................................................................... 10
3.2. Key Reasons for Learning Programming ................................................................... 11
3.2.1. Challenges in Understanding the Context of Conversations ............................... 11
3.2.2. Challenges in Building Rapport .......................................................................... 12
3.3. Approaches Used for Learning Programming ............................................................ 13
3.3.1. Beginning the Learning Process .......................................................................... 13
3.3.2. Formal and Informal Learning Approaches ......................................................... 14
3.4. Perceptions of the Learning Process and Feelings of Failure .................................... 15
3.4.1. Takes Too Much Time ......................................................................................... 16
3.4.2. Too Much Focus on Syntax and Logic ................................................................. 17
3.4.3. Explanations Are Not Relevant .......................................................................... 18
3.4.4. Difficult to Assess the Content’s Reliability ......................................................... 19
3.4.5. Feelings of Social Isolation .................................................................................. 20
3.4.6. Easy to Forget Details Without a Direct Application ........................................... 21
3.5. The Paradox of Learning Programming ..................................................................... 22
3.6. Summary .................................................................................................................. 24

Chapter 4. Investigating Community-Curated Explanations of Technical Concepts for Conversational Programmers ......................................................... 25
4.1. Research Approach .................................................................................................. 25
4.2. JargonAid – a Community-Curated Dictionary of Technical Concepts ................. 26
Chapter 5. Discussion ........................................................................................................... 46
5.1. Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers .................................................................................................................. 46
   5.1.1. Is programming knowledge even necessary? ........................................................................ 47
   5.1.2. Is my chosen learning resource even appropriate? ............................................................ 47
5.2. Design Opportunities for Supporting Conversational Programmers ............................ 48
   5.2.1. Facilitating Discovery of Relevant and Reliable Content .................................................. 48
   5.2.2. Explaining Concepts without Syntax and Logic ............................................................... 49
   5.2.3. Generating Executive Summaries and Visual Explorations ........................................... 50
   5.2.4. Building Conversational Programmers’ Own Communities ........................................... 51
   5.2.5. Towards Structured Community-Curated Explanations ............................................... 52
   5.2.6. Applying Community-Curation in Other Domains ......................................................... 52
5.3. Limitations and Future Work ......................................................................................... 53

Chapter 6. Conclusion ........................................................................................................... 55

References ............................................................................................................................ 56
List of Tables

Table 1. Our participants from local companies and educational and non-profit institutions represented a diverse range of occupations............................ 9
Table 2. Formal and informal resources used by participants .................................................. 14
Table 3. Six common reasons for feelings of failure among conversational programmers when using modern resources .......................................................... 17
Table 4. List of Big Data Related Concepts ........................................................................... 31
Table 5. Descriptive Statistics for Evaluation Questionnaires ................................................. 39
List of Figures

Figure 1. The Overview of JargonAid Interface ................................................................. 27
Figure 2. JargonAid allows users to add short explanations with images .......................... 28
Figure 3. Example Conversations ....................................................................................... 29
Figure 4. When logged in as a study participant, certain changes were made to the JargonAid system ........................................................................................................... 34
Figure 5. Evaluating Users’ Familiarity with the Concepts ................................................. 35
Figure 6. Evaluating Users’ Subjective Feelings ................................................................. 36
Chapter 1.

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], older adults [22], and non-native English speakers [21]. 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]. While there are many different motivations and approaches towards developing general technical literacy [27], in this case, learners are described as conversational programmers as they seek to build specific programming literacy 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 thesis, 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, 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 re-sources 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.

As a step towards addressing this mismatch, we explored how conversational programmers could potentially benefit from community-curated explanations for
technical concepts that minimize jargon and remove programming syntax and logic details. We first designed JargonAid, a community-curated online dictionary that allows users to add simple explanations and example conversations for technical concepts (Figure 1). For our evaluation, we compared conversational programmers’ overall perceptions in pursuing technical conversations after interacting with community-curated explanations vs. formal explanations (e.g., from textbooks, online documentation). Our key findings with 10 conversational programmers showed that they perceived community-curated explanations to be easier to understand, maintain focus, and made them more confident about participating in follow-up conversations.

In our discussion, we reflect on the key insights from this thesis and how human-computer interaction and computing education can play a key role in better understanding and supporting conversational programmers in the future. Furthermore, we also reflect on the value of community-curated explanations for conversational programmers and weigh in on the advantages and disadvantages of the approach introduced in JargonAid.

The main contributions of this thesis are:

- providing empirical evidence characterizing the unique learning needs of conversational programmers and how these needs differ from populations of end-user programmers and professional programmers

- illustrating how modern learning resources that focus on artifact-creation can fail the needs of conversational programmers

- presenting the design of JargonAid and providing initial insights into how conversational programmers can benefit from community-curated explanations when trying to understand technical concepts
Chapter 2.

Related Work

This thesis 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), how people interact with formal and informal programming learning environments and practices of professional-oriented online communities.

2.1. 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 [49], 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 [12,13].

Recent studies show that another emerging non-traditional learner population consists of conversational programmers [6,7]. 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 [45,46], and they share some similarities with conversational programmers. Although teachers may never need to write code on-the-job [45], 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 work found similar sentiments amongst conversational programmers.

2.2. 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 [53]. Research on non-CS major students taking intro CS courses [6,18,56] 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 [23], some recent work has explored formal ways of making programming relevant for non-CS students [18,19,22,24,41]. For example, efforts have been made to teach programming skills in the context of media computation [22,24], 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 [16,60]. 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 work 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.
2.3. 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 [43,53]. 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 [26,31,36]. 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 work 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 [2,38]. These forums effectively facilitate discussion and peer-to-peer knowledge exchange among learners writing code [38,48,51]. 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. In addition, we found that conversational programmers often felt like “outsiders” in communities targeting artifact creation needs. Furthermore, we conducted a comparative study to investigate how resources designed for communication needs have different impacts on conversational programmers compared to the resources targeting artifact creation.

2.4. Professionally-oriented Online Communities

People who have common goals, shared expertise and passion often participate in special-interest groups to support learning and knowledge exchange, which is
commonly described as *communities of practice* (CoPs). According to Wenger[57], community of practice follows three characteristics, 1) an identity defined by a shared domain or interest, 2) the members interact and learn together through joint activities and discussions, helping each other and sharing information, 3) members of a community of practice develop a shared practice.

Studies have examined how community identity plays an important role in fostering sense of community, facilitating professional development and providing social support among practitioners in different settings, for example, graphic design community [42], makerspaces [28], fanfiction communities [14], programmers [17,40]. In programming settings, *Stack Overflow* fits well within the community of practice framework. *Stack Overflow* allows users to search for answers for programming-specific problems and post their own questions to the community that can be answered by other members in the community. Our second study explored the benefits and drawback of building conversational programmers’ own community to provide explanations of technical concepts and how the explanations curated by the community differ from formal explanations retrieved from professional programmers’ community. Furthermore, our second study pointed the social value of building conversational programmers’ community, for example, enhancing online professional reputation, socializing with other people in the industry, and potential career opportunities.
Chapter 3.

Understanding Learning Strategies of Conversational Programmers

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.

3.1. Research Approach and Method

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

3.1.1. 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 from local companies and educational and non-profit institutions represented a diverse range of occupations, 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.

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

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

3.1.2. 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.

### 3.1.3. Data Analysis

We transcribed the audio recordings and did an open coding of the data using *ATLAS.ti*. We used an inductive analysis [11] 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 overlapping 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.

### 3.1.4. 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.

3.2. **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 [6,7] 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.

3.2.1. **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)

3.2.2. 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.

3.3. 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.

3.3.1. 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)

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”.

### 3.3.2. 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.

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

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

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 *Codacadey, 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.

**3.4. 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).

**3.4.1. 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 [23,60]). 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)
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

3.4.2. 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)

3.4.3. 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 [25,31,54], 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.

3.4.4. 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 [2,13,30]. 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)

3.4.5. 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)

3.4.6. 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).

3.5. 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 these Facebook groups and LinkedIn groups”. (P2)

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.

3.6. Summary

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.
Chapter 4.

Investigating Community-Curated Explanations of Technical Concepts for Conversational Programmers

In this chapter, we explore how conversational programmers could benefit from community-curated explanations. We designed JargonAid, a community-curated online dictionary for technical concepts where authors can add technical explanations and examples without using jargon. For our evaluation, we carried out a comparative study with 10 conversational programmers and investigated participants’ perceptions of community-curated explanations vs. explanations retrieved from formal learning resources. We discuss the advantages and disadvantages of community-curated explanations and the potential to extend JargonAid as a collection of both formal explanations and community-curated explanations.

4.1. Research Approach

As suggested in the previous chapter, conversational programmers develop a sense of failure when using modern learning resources that contain programming syntax and logic details, technical jargon, and only focus on building artifacts. Motivated by these findings, we explored the following research questions in the next part of our research:

**RQ1:** To what extent would conversational programmers feel confident to talk about technical concepts in technical conversations if they could access community-curated explanations that did not use any code-level details (syntax and logic)?

**RQ2:** What would be the advantages and disadvantages of providing conversational programmers with community-curated explanations and conversation-specific examples of technical concepts?

To answer these questions, we used a user-centered design process and developed JargonAid, a community-curated online dictionary that encourages conversational programmers (and other users) to add explanations for technical concepts without using jargon. Users can add short explanations either retrieved from a
formal learning resource (e.g., online encyclopedia, articles, or textbooks), or describe the concepts in their own words. Although the primary form of providing explanations in JargonAid is text-based, users can also add figures and links to interactive media. Our design also introduces a dialogue-based mechanism such that a user can explain a technical concept by showing how it is being used in a real-world conversation. Finally, JargonAid incorporates other features such as voting, commenting, and wiki-style editing. A detailed description of the tool is provided in section 4.2.

Next, we carried out a comparative study with 10 participants who fit the definition of conversational programmers to compare community-curated explanations in JargonAid with formal explanations. Each participant was given 6 technical concepts and the task was to use JargonAid to facilitate their understanding of the given concept. For each concept, we randomly showed them either formal explanations (control condition) or community-curated explanations (experimental condition). We collected data from many sources: questionnaires, think-aloud, observations, interviews and qualitative analysis of the explanations written by participants. A detailed description of the study protocol and data analysis is provided in section 4.3.

4.2. JargonAid – a Community-Curated Dictionary of Technical Concepts

This section introduces JargonAid, a community-curated online dictionary that allows conversational programmers to add explanations for technical concepts.

4.2.1. Design Goals

To inform the design of JargonAid, we considered the following problem scenario:

Consider Alex, a new business intern who has no computer science background and is working on a project focused on big data technologies with a development team. Alex is preparing for an internal meeting about the launch of a new data analytics tool with the business team and the development team. He wants to warm up with several concepts so that he can follow the technical conversation and participate in the discussion.
To help conversational programmers like Alex to quickly get familiar with technical concepts and be prepared for using the concepts in conversations, we explored the design of JargonAid (Figure 1), a community-curated online dictionary that allows users to easily search and add explanations. Our design of JargonAid was motivated by the following four key design goals:

1. **Facilitating Discovery of Relevant and Reliable Explanations**

   In our interview study, we found that conversational programmers faced challenges in sifting through irrelevant search results when looking up explanations of technical terminologies. One goal of JargonAid was to use information curation practices so that users can add only the relevant details to explain a concept and provide additional information as needed (e.g., through images, links, etc.).

2. **Explaining Concepts Without Syntax and Logic**

   Given conversational programmers’ unique learning needs and lack of need to build artifacts, JargonAid encourages technical explanations to be concise and easily “skimmable” in a few minutes. We specifically added instructions for adding such explanations (Figure 2). For example, we reminded contributors that the target users of
JargonAid are non-technical people and that explanations should avoid jargon. We encourage contributors to use real-world examples and show how technical concepts would be used in conversations. We also limited the length of explanation to 150 words to keep them concise.

![MapReduce](image)

**Figure 2.** JargonAid allows users to add short explanations with images

### 3. Building Conversational Programmers’ Own Communities

Our previous study pointed out that conversational programmers’ general perceptions of forums like “Stack Overflow” were negative and they felt like an outsider when using such resources. We tackled this issue by incorporating social features such as voting and commenting so that conversational programmers can evaluate the relevance of explanations from the perspective of their own domain and learning needs. In addition, contributors are encouraged to edit their profile with a short bio that would be displayed with the explanations they have contributed.
4. Demonstrating How to Use the Concept in a Technical Conversation

Given that conversational programmers’ main goal is to engage effectively in technical conversations, JargonAid directly provides them with examples of using a technical concept in a real-world development scenario. The example in a conversation component consists of the subject of the conversation, a background of the characters and an interactive dialogue frame displaying the content of the conversation (Figure 3).

Figure 3. Example Conversations

4.2.2. Implementation

We implemented JargonAid using standard web technologies: HTML5, CSS3, and JavaScript. The majority of the JargonAid system is written in client-side JavaScript and is split into numerous UI components managed by React.js, a declarative JavaScript library for building user interface. All the client-side resources are compiled and managed by Webpack, a JavaScript module bundler for packaging assets. As there are no client-side requirements, JargonAid can be run on any computer with a modern, standards-compliant web browser and a mouse. JargonAid has been tested in Chrome 67, Safari 11.1.2, Firefox 61.0.1, Internet Explorer 11.1. and Edge 42.

The server-side of JargonAid is written in NodeJS using the Express web framework. The server handles functions such as user account management, saving and retrieving explanations from the connected MongoDB database.
4.3. Study Design and Analysis

We carried out a comparative study with 10 participants who fit the definition of conversational programmers to compare community-curated explanations in JargonAid with formal explanations taken from existing online resources. In this section, we present the detailed study protocol and data analysis.

4.3.1. Setup Stage: Adding Content to JargonAid

In this section, we explain how content was generated and added to JargonAid for the comparative study. A list of big data related concepts was selected by combining suggestions from experts and popular press. Followed by that, formal explanations were retrieved manually from Wikipedia and textbooks by the primary researcher. Finally, we recruited 6 participants as contributors to simulate community-curated content. By the end, each concept had at least 6 explanations (2 formal explanations retrieved by researchers and 4 community-curated explanations written by contributors in their own words) and 1 example (created by contributors based on their personal experience) showing how it was used in technical conversations.

Picking Big Data Related Concepts

We first consulted domain experts in a data science graduate program and selected 12 concepts that were frequently used in technical conversations. Big data-related concepts such as MapReduce, Database were frequently mentioned by conversational programmers in the previous study so we decided to narrow down the topic to “big data” and we came up with a typical problem scenario that was described in 4.2.1. We then came up with an initial list of 24 big data-related concepts (Table 4) by integrating the ones that were frequently mentioned on business blogs¹ ². Then we invited 6 students from a data science graduate program to vote on the concepts which they had frequently discussed in technical conversations with non-computer scientists.

Based on the voting, the following 12 concepts in Table 4 were tested in the control experiment.

<table>
<thead>
<tr>
<th>Term (picked for the study)</th>
<th>Number</th>
<th>Term (picked for the study)</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Mining</td>
<td>6</td>
<td>Clustering Analysis</td>
<td>4</td>
</tr>
<tr>
<td>Data Visualization</td>
<td>6</td>
<td>Distributed File System</td>
<td>4</td>
</tr>
<tr>
<td>Database</td>
<td>6</td>
<td>IoT</td>
<td>4</td>
</tr>
<tr>
<td>Hadoop</td>
<td>6</td>
<td>Load Balancing</td>
<td>4</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>6</td>
<td>R</td>
<td>4</td>
</tr>
<tr>
<td>MapReduce</td>
<td>6</td>
<td>Structured Data</td>
<td>4</td>
</tr>
<tr>
<td>Neural Network</td>
<td>6</td>
<td>Data Aggregation</td>
<td>3</td>
</tr>
<tr>
<td>NoSQL</td>
<td>6</td>
<td>ETL</td>
<td>3</td>
</tr>
<tr>
<td>Spark</td>
<td>6</td>
<td>Metadata</td>
<td>3</td>
</tr>
<tr>
<td>SQL</td>
<td>6</td>
<td>Petabyte</td>
<td>3</td>
</tr>
<tr>
<td>Classification Analysis</td>
<td>5</td>
<td>Database Administrator</td>
<td>2</td>
</tr>
<tr>
<td>Cloud Computing</td>
<td>4</td>
<td>Predictive Analytics</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4. List of Big Data Related Concepts

**Adding Content from Formal Resources**

Next, we came up with at least 2 formal explanations for each concept from formal resources. According to the manual of Wikipedia, the first section\(^3\) (also known as lead or introduction) stands on its own as a concise overview of the article’s topic so the researchers took the lead section of a Wikipedia article as a formal explanation. In addition, we also visited the glossary table from textbooks\(^4,5\), slides from popular MOOCs\(^6\), authoritative technical blog posts\(^7\) to get the short definition of the concepts.

---


\(^6\) [edX: Cloud Computing](https://www.edx.org/edxcloudcomputing)

Simulating Community-Curated Content

Next, we invited 6 additional graduate students to be contributors to construct and vote for explanations for the selected terminologies. Contributors were brought to the lab and first encouraged to put themselves in the shoes of conversational programmers and browse, vote and comment on existing content.

Then contributors were asked to come up with explanations and example conversations using their own words. We gave contributors suggestions such as avoiding introducing new jargon, using real-world examples or checking explanations on Wikipedia/textbooks/blog posts for inspiration to help them better construct the explanations.

To help contributors construct the example conversations, we encouraged them to recall a recent conversation with colleagues/friends and provided them a list of potential conversational subjects to consider (e.g., the pros and cons of using certain technology, comparison of multiple terminologies, cost/benefit trade-offs for certain technology, how/when the technology is used in real development scenario). In addition, we provided contributors the following scenario to give them a better sense of the target users of the dictionary:

Suppose you are Bob, a senior marketing manager working at an e-commerce company and you are trying to explain some technical jargons through JargonAid to Alice, the business intern who has no computer science background. Alice is preparing for an internal meeting with business team and development team. She wants to warm herself up with several concepts so that she could follow the technical conversation and participate in the discussion.

A pre-test questionnaire and post-test interview were conducted on contributors to collect their feedback on the system’s usability and their perceptions on how the tool could work in practice. Contributors are rewarded a $15 CDN Amazon gift card in return for their participation.

The raw content input by contributors was further fine-tuned and filtered by the researcher and 4 additional experts who were familiar with conversational programmers’ unique learning needs. By the end, each concept had at least 6 explanations (2 formal explanations and 4 community-curated explanations) and 1 example showing how it was used in technical conversations.
4.3.2. Comparative Study with Conversational Programmers

We compared community-curated explanations in JargonAid with formal explanations (e.g., from textbooks, online documentation) through an-hour-long study with 10 participants who fit the definition of conversational programmers. As explained in Figure 4, after the data input step, we modified the JargonAid user interface such that participants who were asked to browse through different concepts would see corresponding explanations from the community (experimental condition) or formal resources (control condition). The explanations were shown in a different random order to each participant and the source of the explanation was hidden in both conditions.

Control Condition (CC): Participants were able to see only the top 2 formal explanations.

Experimental Condition (EC): Participants were able to see only the top 2 community-curated explanations (that included example conversations).

Each participant was given 6 concepts (3 in control condition and 3 in experimental condition) and the task was to use JargonAid to facilitate their understanding of the given concept. Before the study, we asked participants’ familiarity with the concepts by ranking the following statements on a 5 Likert Scale (1 as strongly disagree and 5 as strongly agree), which was listed in Figure 5.
Figure 4. When logged in as a study participant, certain changes were made to the JargonAid system

1. Normal View

   ![Normal View Diagram]

   **Normal View**: When logged in as a normal user or contributor, the user will see the complete interface with source of the explanation and options to add a new explanation.

2. Control Condition

   ![Control Condition Diagram]

   **Control Condition**: When logged in as a study participant and assigned control condition for the given concept, the participant will only see 2 explanations retrieved from formal resources. However, the participant would not be notified the source of explanations during the experiment phase.
3) **Experimental Condition:** When logged in as a study participant and assigned experimental condition for the given concept, the participant will only see 2 explanations and 1 example conversation written from domain experts. However, the participant would also

<table>
<thead>
<tr>
<th>Subject</th>
<th>What is data mining?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>Bob</td>
</tr>
</tbody>
</table>

### Before checking the explanations, please rank the following statement on a scale of 5

<table>
<thead>
<tr>
<th>Statement</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>I have heard about / read across this concept before.</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>I already have a depth understanding of this concept and feel confident to talk about this concept.</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Figure 5. Evaluating Users’ Familiarity with the Concepts**

Participants had up to 3 minutes to check the JargonAid dictionary and after that, we asked participants to hide the browser and write down the explanation of the concept in their own words on paper. We developed questions to measure participants’ perceptions and subjective feelings on the explanations after they finished checking explanations for each concept. A detailed evaluation matrix is listed below:
Participants were required to fill the pre-test questionnaire which gathered some demographic information like age, gender, and education background. We also carried out a semi-structured post-test interview which probed into their general perceptions on the community-curated explanations. Each semi-structured post-test interview session lasted around 10-15 minutes and the whole study lasted around 45-60 minutes in total. We began the interview by asking how they perceived elements within the explanation, such as the related images, community suggestions (upvote/downvote numbers and comments), length of the explanation, and examples of how the concept is used in a technical conversation. Next, we explained the purpose of the comparative study, demonstrated the normal view of JargonAid which displays explanations with its information source (as shown in Figure 4. When logged in as a study participant, certain changes were made to the JargonAid system) and asked participants to reflect back on the tool. Lastly, we ended the interview by inviting them to propose features that they would like the tool to have. Each participant was offered a $15 CDN Amazon gift card in return of participation.

Figure 6. Evaluating Users’ Subjective Feelings
4.3.3. Recruitment

Participants were recruited through snowball sampling. We first contacted non-CS students through personal connections and by advertising posters. Next, we selected additional conversational programmers through word-of-mouth, asking each participant to introduce us to other non-CS friends or classmates whom they knew. Our participants had to fit the following criteria to take part in the interviews (based on the definition of conversational programmers introduced earlier):

1) Do not have a formal degree (major or minor) in computer science/engineering/IT-related fields;

2) Are not working in any kind of a software developer/engineer role and are not required to write code as part of your job or course project.

In total, we conducted the control experiment with 10 participants (6 women, 4 men) studying nine different majors at a local university: Accounting (2), Math (1), Economics (1), Statistics (1), Finance (1), Linguistics (1), Molecular Biology (1), Business (1), and Health Science (1), where 2 participants are graduate students and 8 participants are undergraduate students.

4.3.4. Data Analysis

Our evaluation used a mixed-methods approach, combining many data sources: questionnaires, observations, think-aloud, interviews, and qualitative analysis of the explanations written by participants. We first analyzed the evaluation questionnaires to compare results between the two conditions. We also coded the interview transcripts and observational notes to discover recurring themes.

4.4. Key Findings

Overall, participants’ general perceptions of community-curated explanations were positive. Our analysis reveals that conversational programmers found community-curated explanations to be easier to understand, helped them stay focused, and made them feel more confident compared to using formal explanations. We discuss the potential to use community-curated explanations as supplementary material to formal
explanations for facilitating conversational programmers’ understanding of technical concepts. Finally, we compare this combined approach to face-to-face explanation and the general online search strategies.

4.4.1. Baseline Knowledge of Concepts Used in the Study Tasks

Before presenting our main findings on how conversational programmers perceive community-curated explanations, we first provide a brief overview of participants’ initial understanding of the given concepts (Table 4) that were used in the study tasks.

Overall, participants had little previous knowledge of the given concepts in both conditions. When questioned using a 5-point Likert scale (1: strongly disagree, 5: strongly agree), most participants indicated that they had not come across the concept before (mean=2.57, σ=1.38), and even fewer participant felt confident in talking about the concept (mean=1.78, σ=1.11).

4.4.2. Community-Curated Explanations vs. Formal Explanations

After showing participants either formal explanations or community-curated explanations but not revealing the source of explanations, we asked participants 6 evaluation questions (Figure 6) to measure their perceptions. Because this was a questionnaire-based study, all of the questions are self-report, and participants’ responses reflect their subjective feelings, but not actual knowledge. For each question, we asked participants to rate using a 5-point agreement Likert Scale (1 as strongly disagree and 5 as strongly agree). We performed descriptive statistics to observe basic features of the data, as listed in Table 5.

Although Wilcoxon-signed tests showed that none of these differences were significant (perhaps due to our small sample size), the results do reveal some interesting trends. For example, the participants’ confidence levels in using the given concept in a technical conversation appeared to improve in both conditions compared to their initial confidence level before checking the explanations (mean=1.7 in the control condition, mean=1.86 in the experimental condition). In addition, there was an improvement in the participants’ confidence levels when they saw community-curated explanations vs formal
explanations (mean=3.26 in the control condition, mean=3.46 in the experimental condition). Participants found community-curated explanations to be more understandable (mean=3.8 in the control condition, mean=4.13 in the experimental condition) and found it easier to stay engaged (mean=3.9 in the control condition, mean=4.17 in the experimental condition). Interestingly, participants felt that they would still need to seek other resources to fully understand the concept when looking at the community-curated explanations vs. formal explanations (mean=3.97 in the control condition, mean=3.7 in the experimental condition).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Control Condition</th>
<th>Experimental Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>The explanations were easy to understand.</td>
<td>3.80</td>
<td>4.13</td>
</tr>
<tr>
<td>The explanations contained the right amount of detail.</td>
<td>3.77</td>
<td>3.60</td>
</tr>
<tr>
<td>I trust the accuracy of the explanations.</td>
<td>4.33</td>
<td>4.23</td>
</tr>
<tr>
<td>I would still need to look for other resources to fully understand this</td>
<td>4.00</td>
<td>3.70</td>
</tr>
<tr>
<td>concept.</td>
<td>1.40</td>
<td>1.44</td>
</tr>
<tr>
<td>I now feel confident about using this concept in a conversation (e.g.,</td>
<td>3.27</td>
<td>3.47</td>
</tr>
<tr>
<td>with a CS major student).</td>
<td>0.94</td>
<td>1.30</td>
</tr>
<tr>
<td>It is easy for me to stay focus on the explanations.</td>
<td>3.90</td>
<td>4.17</td>
</tr>
</tbody>
</table>

Table 5. Descriptive Statistics for Evaluation Questionnaires

Despite that community-curated explanations seemed to make conversational programmers feel more confident to engage in technical conversations, participants perceived formal explanations to be more detailed than community-curated explanations (mean=3.77 in the control condition, mean=3.6 in the experimental condition), and they also perceived formal explanations to be more accurate (mean=4.33 in the control condition, mean=4.23 in the experimental condition).

In addition to the questionnaire responses, we also further analyzed the post-test interview responses and investigated participants’ perceptions of the two different types of explanations they saw in the user study.
4.4.3. Attributes of a Useful Community-Curated Explanation

In the post-test interview, we invited participants to reflect back on different attributes of the community-curated and formative explanations and what they considered to be useful. We probed into how they perceived different elements within the explanation, such as the related images, community suggestions (upvote/downvote numbers and comments), length of the explanation, and examples of technical conversations. Our key results are discussed below.

Wish to See More Examples of How the Concept is Used in Conversations

On a point-5 Likert Scale, the questionnaire data showed that participants found the dialogue-based example was particularly useful (mean=4.13, 1 as not helpful at all, 5 as extremely helpful) for improving their conceptual understanding and engaging in technical conversations.

Participants further reported that the example conversations made it easier to understand the concept because the language in conversations was more informal. The dialogue-based explanations provided a way for users to see how the concept was applied in real life, as explained by a participant from an Economics background:

It [the example] provides a way to see how it is used in conversation in real life, so it’s more practical. (U4)

In addition, participants mentioned that the Q&A form of the dialogue helped them maintain focus, and more importantly, inspired them to think and digest the content. For example, a participant from Molecular Biology found the example conversation about “Hadoop” relevant to her confusion after reading the short definition:

When I was reading the text-based explanations, I just passively went through it. But for the dialogue-based conversation, the questions asked by the non-technical people happened to be the questions I was thinking about. I felt motivated to read and digest the dialogue. (U8)

Helpfulness of Images

Participants found images corresponding to explanations only helpful when they were related to the text explanations directly. And, the participants also noticed that images worked for some concepts, but not all of them. For example, participants pointed
out that images facilitated their understanding of terms like map reduce, data visualization and classification analysis, but not for terms like Hadoop, or SQL.

Images were perceived to be helpful in remembering the concepts better. One participant from Linguistics reported that sometimes people remembered vocabularies by recalling visual elements when learning a foreign language, which he found to be similar to learning a technical concept:

I find myself remembering images better than text... As a kid, we practice English vocabulary by flash cards with images. So, if the text explanation is connected to the image, then it helps me remember the explanations and the concept. (U7)

In addition to looking at images, participants expressed a general preference for seeing explanations more visually. When asked about what other features they would like to see in the system, several participants mentioned that they wanted to see short video/animated pictures/graphs/doodles that explain the given concept.

**Community Suggestions as a Reference**

The explanations on JargonAid can be voted and commented upon by contributors. When asking about whether community suggestions such as voting or commenting affect their perceptions of the explanations, participants stated that the community suggestions were helpful, especially when they were not familiar with the concept at all. But when looking for a deeper understanding, most participants mentioned that they would still want to read all the explanations:

I will still read explanations if it has many downvotes... just to see the difference between multiple explanations and to understand things from different perspectives. (U1)

**Trade-off between Accuracy, Length, and Ability to Focus**

Participants’ responses revealed two different attitudes towards the length of the explanations. Some participants preferred shorter explanations with only 1-2 sentences:

I found the shorter ones are clearer and more readable. I prefer explanations in simple sentence structure, for example, by just saying X is what... (U9)

Other participants preferred longer explanations that contained more detail and examples, with analogies to topics that conversational programmers are already familiar
with in their domain. It was a trade-off between accuracy and focus, as summarized by one participant from accounting:

If it’s too long, it’s hard for readers to concentrate. But if it’s too short it loses the details. It’s like a trade-off between accuracy and focus. (U3)

4.4.4. Combining Community-curated Explanations with Formal Explanations

After evaluating different attributes of the explanations in the post-test interview, we probed into participants' perceptions of the source of the explanation (given that it was hidden during the study).

Our previous study showed that modern learning resources were perceived to contain more jargon and were not helpful for conversational programmers. By introducing community-curated explanations, our initial results show that participants appreciated the value of community-curated explanations written without jargon (and found them to be more useful than the formal explanations, even when the source was unknown). One participant from economics said that the community-curated explanations were unique because they combined other people’s experience, “Here, it’s about other people’s experience, so it’s different from just a standard explanation provided by Wiki. (U8)”. Another participant from business mentioned that community-curated explanation could be useful as there are different communication styles used by “business people” and “computer science people”:

I guess for business people...we are trying to communicate stuff in a simpler way, in a way that general people will understand. But for programmers, they are already educated in computer science for so many years, and they have so much background knowledge, sometimes they probably not even realize it themselves, but they are using so many words that people who are not from tech background would understand. (U3)

One drawback of community-curated explanations noted by participants had to do with the community-curated explanations being somewhat less trustworthy since the source was unknown. To increase their trust, some participants suggested to add more information about the authors (which is the long-term goal of JargonAid):

I wish this system could connect to websites like LinkedIn, just to provide me more background of the other users. (U6)
As a result, it was not the case that participants would totally give up on explanations from Wikipedia and textbooks. Most participants mentioned that they would still like to read both formal explanations and community-curated explanations, where the formal explanations gave them an overall introduction to the concept and the community-curated explanations served as a supplement. For example, one participant from accounting explained the following:

I will probably read explanations from Wikipedia first, I know it’s not the best explanation, but at least it gives you a good introduction to understand something...But, then I will read other people’s interpretations, the more I read, the better idea I will have. Because people may explain it from other perspectives. (U1)

4.4.5. Perceptions about JargonAid Use in Practice

In the pre-test questionnaire, when asked about their past experience to understand a computing science concept, most participants mentioned searching on Google and asking friends. In our interviews, we further probed into how participants perceived JargonAid in comparison to searching online and/or asking friends. We also describe how participants would use JargonAid in practice (if it were available).

JargonAid provided conversational programmers explanations from different perspectives, where the curation part played an important role in improving efficiency for conversational programmers to find appropriate learning resources. On a scale of 5 (1 as not at all, 5 as definitely agree), participants felt not so confident (mean=2.8) with using Google to search for explanations of technical concepts. “You sometimes get what you want sometimes not. (U6)”, as reported by a Ph.D. student from finance.

Participants also complained that sometimes asking friends was not helpful, especially a technical friend – “Asking him (a computer science student) is even worse. He didn’t give me any confidence that I could figure out the concept, it brings me panic. (U3)”. Besides the uncomfortable and stressful feelings of asking friends, one participant mentioned that he could have a better understanding by just reading through different interpretations:

Compared to asking in person, the person may not be well-prepared to give you a good example... but, on the system, I might have a better understanding and I could read through different examples. (U1)
In addition, participants felt that they had the freedom to browse and revisit the explanations, which helped them remember better. A participant from health sciences background stated:

On the system, I could read and retain it in my memory. But if my friend explained to me, it kind of strayed in my mind and I will probably forget about it right away...and, I probably wouldn’t like to bother my friend again. (U10)

On the other side, talking to friends face-to-face provided conversational programmers opportunities to ask project-specific questions. One participant from molecular biology explained the advantages of asking friends in person, “Because it’s face to face, you could always ask follow-up questions, and really dig in deep into the matter. (U8)”. Some participants further suggested us to improve the design of JargonAid by having synchronous communication feature where they could have such conversations with the authors of the explanations.

4.4.6. Contributors’ Perspective on JargonAid Use in Practice

Additionally, we analyzed the post-test interview data from the setup stage to investigate how JargonAid would use in practice on contributor’s perspective. In general, contributors found the process of adding explanations and example conversations easy (mean=4.17, on a scale of 5, 1 as extremely difficult, 5 as extremely easy).

When asking about whether they were willing to contribute towards such a system in real life, most contributors expressed a positive attitude. A contributor reflected on his experience of using Quora, a knowledge Q&A site: “I gained social values when I am answering domain-specific questions on Quora. So, I would totally be willing to contribute to this tool (JargonAid) to enhance my online reputation with other business people, who might be my potential customer or collaborator. (C3)”. Others suggested mechanisms such as “give contributors actual money reward, like Amazon Mechanical Turk. (C4)” to motivate contributors to construct high quality explanations.

4.5. Summary

This chapter presented JargonAid, a community-curated online dictionary that allows authors to add simple explanations and example conversations for technical
concepts. For our evaluation, we compared community-curated explanations in JargonAid with formal explanations (e.g., from textbooks, online documentation). We found that conversational programmers perceived community-curated explanations to be easier to understand, allowed them to maintain focus, and made them more confident about participating in follow-up conversations. We discuss the potential of using community-curated explanations as learning tool for conversational programmers and reflect on the advantages and disadvantages of our approach.
Chapter 5.

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 [5,23,60] and prefer informal learning approaches [12,43]. 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” code examples [13], 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 [45], 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 thesis, 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. Furthermore, we explored how conversational programmers could potentially benefit from community-curated explanations for technical concepts that minimize jargon and remove programming syntax and logic details.

We now reflect on the mismatch of expectations that conversational programmers experience and how HCI and learner-centered design [23] approaches can play a pivotal role in better supporting this emerging learner population.

5.1. Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers

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.

5.1.1. 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 [50], people tend to associate anything technical with programming [15]. 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 [8,62].

5.1.2. 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 [63]. 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.

5.2. Design Opportunities for Supporting Conversational Programmers

Here we consider the design implications of our findings from the interview study and comparative study, and how we can better support conversational programmers.

5.2.1. 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, one implication is to look into facilitating discovery of relevant and reliable content. Our second study further shows that conversational programmers could get benefit from such a community-based dictionary that shows curated overviews of the concepts with example conversations that are focused on big data related topics in a business context. In the future, it worth exploring how this method would scale for other technical concepts.

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. Our findings from the second study shed light on how community voting and commenting feature influenced conversational programmers’ judgement on the quality of the learning materials. However, we do not specifically investigate how the contributors’ background (e.g., occupation, level of expertise, working experience, educational background, gender) affected the way conversational programmers perceived these recommendations. There are opportunities for future work to investigate who the relevant community leaders would be and how would they make contributions towards endorsing a particular resource.

### 5.2.2. 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?

With JargonAid, we found that conversational programmers perceived the text-based “skimmable” explanations and interactive examples of conversations to be useful. In particular, the mechanism of teaching conversational programmers how to talk about a particular concept in the context of a real-world development scenario could help participants maintain focus and get inspired to think and digest the content. In addition, participants expressed the wish to see more visual elements in explaining concepts with complicated algorithms or process, which is worthwhile to study in the future.

A popular approach that has been explored in research and practice is the design of novice-friendly “drag-and-drop” [44] programming languages and systems such as *Alice* [10], *Scratch* [47], and *Code.org* [64] to make programming more attractive for children [39,58] and other novices [20]. However, none of our participants in the interview study 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 formal courses with emphasis on more conceptual instruction of computing concepts without writing code [1,19,35,41,56]. 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. JargonAid serves as an initial example of helping conversational programmers see explanations that talk about a particular concept in the context of a real-world development scenario. It may be fruitful to explore how to expand such interactive reference components and other design possibilities for connecting real-world context with programming-related concepts for conversational programmers.

5.2.3. 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. In fact, pros and cons, comparison of two concepts are two of the most commonly used subject when contributors naturally came up with example conversations on JargonAid in the second study (e.g, comparing Spark and Hadoop) and participants found these topics relevant to problems encountered in real life. However, more explorations are needed to determine whether and how such mechanisms could be applied to concepts in domains other than big data (e.g., computer network, web application development, cybersecurity).

At the code level, perhaps there is a need for more visual explorations like interactive neural net explorations [4], *explorable explanations* [65] or algorithm animation [3] 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 [66,67].
5.2.4. 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 [59], CS teachers [46], and even product managers [8]. 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]).

Our experience with designing JargonAid shows that conversational programmers find these suggestions and mentorship from experienced conversational programmers more practical and relevant compared to suggestions from communities where the target users are professional programmers. However, due to the limitation of our second study, we only simulated the community activities by recruiting contributors with technical background and providing them a problem scenario. We believe that more insights will be added if future deployment studies could recruit contributors from experienced conversational programmers, provide participants tasks in a more realistic settings (e.g., provide them a scenario for technical conversation) and further investigate how this tool could work in practice.
5.2.5. Towards Structured Community-Curated Explanations

With our design of JargonAid and the second user study, we found that conversational programmers found the community-curated explanations to be more understandable and easier to stay focus than formal explanations. However, we observed differences in what made the explanations actually useful and how they were perceived by the end users. Some explanations used intuitive metaphors to explain how a technical concept works—for example, one contributor explained the concept of “MapReduce” as “a person wants to classify documents under time limitation, so he divides the job into small pieces and asks certain amount people to complete a piece at a time. (C3)”. Other contributors preferred to explain the concept concisely by giving the simplest definition—for example, another contributor described “MapReduce” as “a system to process data with a parallel, distributed algorithm. (C2)”. We observed that while some participants preferred the intuitive metaphors that encouraged them to think, others enjoyed the shorter, one-sentence “what is” explanations. In future deployments of JargonAid, it would be worth exploring all the frequent mechanisms/writing styles to explain a technical concept and how these mechanisms are perceived by a larger sample of conversational programmers.

The proper mechanism to explain a concept can also vary between different concepts. For example, participants mentioned that images were sometimes helpful in facilitating their understanding of the concepts (e.g., data visualization, database), while sometimes not (e.g., SQL, Hadoop).

Given the nature of these differences between explanations, we believe that there was no perfect formula or universal mechanism to construct the explanation that adapts to all the concepts. It may be worth to categorize the concepts and investigate what could be the best strategy to explain for each category in the future so that contributors could get more detailed instructions on how to generate high-quality explanations.

5.2.6. Applying Community-Curation in Other Domains

Outside the domain of computing science, there are other cases where people may want to build common ground knowledge to understand domain-specific jargon. For
example, patients may want to warm themselves up about medical-related terminologies before seeing a doctor; general audiences may want to know some rules and sports-related terminologies when casually watching a sports event that they are not familiar with; novice video game players may want to get familiar with the game community’s language style to communicate with more expert players; and, there are many other such examples in other domains.

In the above scenarios, newcomers/novices or the general public often feel stressful when faced with unfamiliar terminologies and situations and would need a certain “translator” to explain the concept in a simple and understandable way. For example, studies have looked into self-diagnosis applications that educate the general public about health issues [61]. We believe that the community-curation mechanism of JargonAid could be scaled more broadly for such audiences, e.g., building a medical version of JargonAid to help patients skim overviews of medical concepts and to improve their confidence levels when engaging in conversations with doctors. In the future, it will be worth exploring the advantages and disadvantages of the community-curation mechanism applied to different domains.

5.3. Limitations and Future Work

In the first study, our focus was only on exploring the perceptions and learning strategies of conversational programmers; 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.

A limitation of the second study is that we collected participants’ subjective feelings on community-curated explanations, which lacks conclusive evidence that this mechanism led to meaningful learning improvements. Future work can incorporate analysis of the explanations written by participants and measure their comprehension. Since we used a lab study approach and provided participants with a pre-defined
problem scenario, we were not able to observe patterns of real-world use. We may run large scale deployments in the future to reveal how community-curated explanations can work in the real-world.

More importantly, when we talk about “grounding in communication” [9], 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 [37,52]. 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.
Chapter 6.

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. Built upon the design implications, we presented JargonAid, a community-curated online dictionary that allows conversational programmers to add explanations for technical concepts and we specifically examined how community-curated explanations would help conversational programmers improve their confidence level and build common ground in technical conversations.
References


