Profiling Conversational Programmers at University: Insights into their Motivations and Goals from a Broad Sample of Non-Majors

Jinyoung Hur
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
jhur10@illinois.edu

Kathryn Cunningham
Department of Computer Science
University of Illinois Urbana-Champaign
Urbana, Illinois, USA
katcun@illinois.edu

ABSTRACT

Background and Context. Instruction in most introductory computing courses is typically focused on how to program. However, non-majors who take computing courses have a diverse set of desired endpoints. One group of non-majors are the conversational programmers, who do not want to program in their career but enroll in computing courses to improve their ability to communicate about technical topics and their competitiveness in the job market. Research suggests that these learners need an alternate instructional approach, but so far, conversational programmers in higher educational contexts have only been studied in a limited number of small-scale studies. **Objectives.** To inform curriculum design for conversational programmers at the university level, we (a) examine the prevalence of conversational programmers among non-majors and their characteristics, (b) understand conversational programmers' desired learning goals and classroom activities, and (c) investigate factors associated with these learners' motivation to learn computing. **Methods.** We designed a survey based on Expectancy-Value Theory and prior work about conversational programmers. We collected responses from randomly sampled non-major students at a large public university, and we analyzed the survey data with descriptive and inferential statistics. **Findings.** We found that conversational programmers are the largest proportion of non-majors in our sample, both overall and across historically underrepresented groups in CS. We replicated prior findings of low self-efficacy for programming of conversational programmers. We found that conversational programmers' motivation for taking more computing courses is paradoxically driven more by their interest in computing than its utility, despite their general lack of enjoyment in computing. We validate a previously proposed set of conversational programmers' learning goals and show that they value employment-oriented learning goals over those focused on conversations. **Implications.** Our results suggest that addressing the needs of conversational programmers can contribute to broadening participation in computing. Our study motivates a learner-centered curriculum design that could address conversational programmers' learning needs by enhancing their self-efficacy and interests prior to focusing on conversational goals.

ICER '24 Vol. 1, August 13–15, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0475-8/24/08
https://doi.org/10.1145/3632620.3671123

CCS CONCEPTS

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

KEYWORDS

Conversational Programmers, Non-majors, Learning Goals, Learner-centered Design

ACM Reference Format:
Jinyoung Hur and Kathryn Cunningham. 2024. Profiling Conversational Programmers at University: Insights into their Motivations and Goals from a Broad Sample of Non-Majors. In *ACM Conference on International Computing Education Research V.1 (ICER '24 Vol. 1), August 13–15, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, page 19 pages. https://doi.org/10.1145/3632620.3671123

1 INTRODUCTION

Introductory computer science (CS) courses have traditionally emphasized writing programs, driven by the assumption that students will need to write code in their future careers [19]. While this instructional approach aligns with the goals of learners aspiring to become professional or end-user programmers, it may not accommodate the learning needs of certain students, notably conversational programmers [45]. Conversational programmers are an emerging subpopulation of non-CS major students (henceforth, non-majors) who learn computing without the intention to write programs [6]. Instead, they are known to value alternative applications of computing knowledge, such as improving communication with technical peers or enhancing job marketability through familiarity with recent technologies [6, 10, 45]. However, these learning goals are outside the primary scope of the conventional introductory CS courses (CS1) this population typically enrolls in, so meeting conversational programmers' needs may require further attention from educators and instructional designers.

While studies have been conducted on conversational programmers, researchers have yet to investigate how common conversational programmers are in the university context, particularly among various demographics and majors. Considering the more diverse backgrounds of non-majors compared to CS majors [38], the undergraduate population of conversational programmers may also consist of students of diverse backgrounds, especially of backgrounds traditionally underrepresented in CS. Exploring the relationship between demographic characteristics and types of desired computing endpoints, therefore, would help us understand whether addressing conversational programmers' needs can broaden participation and increase diversity in computing—a challenge that the field of CS has struggled with [34]. Moreover, previous research on undergraduate conversational programmers focused on only a few

majors, such as Management Engineering [6] and Information [9], leaving uncertainty about the presence of conversational programmers in other majors. Gaining a more comprehensive understanding of how common conversational programmers are across various demographics and majors would help better assess the necessity of designing curricula tailored to meet their needs.

Another research gap in the current literature on conversational programmers is that the extent to which the findings of the studies performed so far apply to a broad population of undergraduate conversational programmers remains uncertain. Previous studies have examined undergraduate conversational programmers' learning goals and expectation for success in programming, however on a small scale [6, 9]. Other works have studied conversational programmers in non-university contexts, such as in the software industry [7] or among adult learners [45]. Consequently, we are yet to have a comprehensive understanding of conversational programmers' learning needs and challenges across the relevant undergraduate population.

As we move towards a learner-centered design process [24, 42] for curriculum that meets the needs of conversational programmers, it is crucial to understand the factors that influence the motivation of this population, such as their prior computing experience and reasons for valuing computing. In addition, it is important to understand how and if they differ from other populations of non-majors, particularly end-user programmers.

In this study, we seek to address these gaps in research on conversational programmers at the university level by conducting a large-scale survey designed based on the framework of Expectancy-Value Theory [15]. Specifically, our research questions are as follows:

- *RQ1: How common are conversational programmers in the population of non-majors that take computing courses?*
- *RQ2: Is the endpoint of conversational programming more popular than other endpoints among certain majors or groups underrepresented in computing?*
- *RQ3: Do conversational programmers have less prior exposure to computing or more negative affective memories about their prior computing experience compared to end-user programmers?*
- *RQ4: Do conversational programmers have different types of subjective task value for computing compared to end-user programmers?*
- *RQ5: Do undergraduate conversational programmers prefer learning activities that focus on code, or on conversation?*
- *RQ6: Do earlier findings about conversational programmers (i.e., expectation of success, learning goals, types of knowledge) generalize across the undergraduate student population of conversational programmers?*

2 BACKGROUND AND RELATED WORK

2.1 Non-majors

Non-majors are a significant and growing audience of computing courses. The 2017 "Generation CS" report found that not only has the enrollment of CS majors grown to unprecedented numbers, but "the number of nonmajors in computing courses increased at a rate equal to or greater than the increase in majors" [3]. The report found that this growth in non-majors enrollment is not limited to the courses designed for non-majors, or even CS1, but is also seen in mid-level and even upper-level electives.

Empirical evidence suggests that the characteristics of non-majors are different from those of CS majors. Prior research from the Building, Recruiting, and Inclusion for Diversity (BRAID) institutions suggests that demographics of non-majors enrolled in introductory computing courses consist of more female students compared to CS major students [38, 39]. At PhD-granting institutions, the introductory computing courses for non-majors has seen higher growth in the proportion of students from underrepresented racial backgrounds than for-majors CS1 [4]. Non-majors in computing courses also tend to be "latecomers," who take their first college-level computing course in the latter half of their undergraduate degree [30, 39]. This timeline means that non-majors often simply have less time to learn computing. Moreover, non-majors tend to have less computing experience before college [39], and they also exhibit lower self-efficacy in computing than majors [33]. Non-majors in computing courses have learning objectives that are different from those of CS-majors: non-majors prioritize learning about computational tools and concepts over learning about programming language and implementation [27].

Efforts have been made to design new computing courses that better accommodate the diverse learning needs and specific challenges of non-majors. For instance, few curricula have been developed to integrate computing practices into other subjects or majors. One notable example is the Harvey Mudd College's "BioComp" course designed for undergraduate biology students to teach computing skills with applications to biology [11, 32]. Additionally, to establish the Program for Computing in the Arts and Sciences (PCAS) at the University of Michigan, a task force engaged with liberal arts and sciences (LAS) faculty and students to define their computing education needs. The task force identified three key themes of computing use for LAS: computing for discovery, computing for expression, and computing for justice [25]. Moreover, researchers at the Georgia Institute of Technology developed contextualized computing courses, including Media Computation (MediaComp) [21], with the goal of motivating non-majors with evidence that computing is useful for something tangible [22]. Researchers at Berkeley developed the Beauty and Joy of Computing (BJC) curriculum, where non-majors work towards computational thinking goals, create art with code, and critique current technologies [20].

A common feature of non-major computing courses are highly scaffolded programming environments that minimize the need for syntactic knowledge, like BJC's use of block-based programming in Snap!, or MediaComp's use of JES, a custom IDE with built-in media manipulation functions [21]. Such approaches are not representative of the real work of non-software developers who program [26], but, the MediaComp approach was successful in reducing students' likelihood of disliking CS1 content [23] and in increasing the retention rate (i.e., lower drop or failure rate) of non-major students [23, 36]. However, MediaComp did not seem to encourage non-majors to change their educational plans to further pursue computing [23].

2.2 End-user programmers

End-user programmers were among the earliest subpopulations of programmers with an endpoint besides software development to receive attention in the computing education and human–computer interaction (HCI) literature. Defined as individuals who engage in programming activities to support their needs, goals, or work other than software development [29, 35], end-user programmers are estimated to be the largest group of individuals who engage in programming activities, surpassing the number of professional programmers [40]. Example career profiles for end-user programmers include data analyst, data scientist, and business analyst [9]. Numerous studies have investigated end-user programmers' perceptions and attitudes toward programming in various subject domains, finding that their value for software engineering practices like testing was lower than their value for domain-specific concerns [29]. Considerable effort has been expended to better support end-user programmers, such as to enhance the accessibility of programming tools for their needs (e.g., programming by example [31] or task-specific languages [5]). In studies of largely self-taught web and graphic designers, Dorn found that these end-user programmers didn't value formal computing education, finding it too focused on syntax rather than concepts [13]. These learners most valued computing knowledge that was relevant to their task at hand [14]. These findings inspired a novel curriculum design focused on embedding computer science knowledge in case studies and examples applicable to learners' needs [12].

2.3 Conversational programmers

Another subgroup of learners with a different endpoint, termed conversational programmers, was first identified by Chilana et al. in a study of management engineering undergraduate students [6]. The conversational programmers they profiled differed from end-user programmers as their goal in learning programming was to be able to speak in the "programmers' language" and communicate effectively with technical colleagues rather than to write code themselves [6]. Further studies confirmed the presence of conversational programmers in various industries and careers. For example, conversational programmers were found in the software industry, seeking to enhance their job marketability by improving conversational skills and gaining a high-level understanding of programming [7]. Some example profiles of conversational programmers' careers include roles in marketing, digital strategy, sales, management, user experience design, and customer relations (e.g., product manager, visual designer, advertising manager, marketing coordinator, entrepreneur) [7, 9, 45]. Subsequently, Wang et al. conducted interviews with adult, employed conversational programmers who were learning about computing to investigate their recent experiences and learning needs [45]. This study concluded that available learning resources in programming were largely inadequate for conversational programmers' learning, leading to feelings of failure due to the mismatch between their learning needs and the resources' excessive focus on programming syntax and logic, or explanations irrelevant to their learning goals.

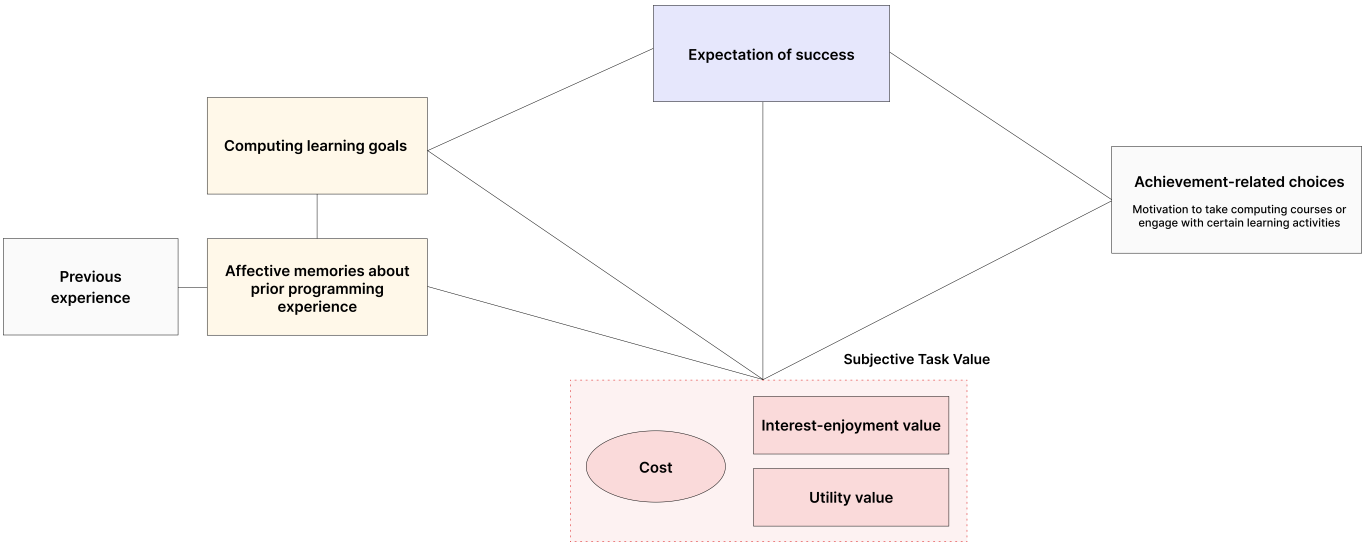
A limited number of attempts have been made to design learning environments specifically for students with conversational programming learning goals. Wang and Chilana developed JargonLite, an interactive dictionary that defines programming concepts and jargon and demonstrates their usage in everyday dialogue [44]. Cunningham et al. proposed Purpose-first Programming, an approach that supports students to learn about domain-specific programming plans so they can focus on the purpose of code without fixating on the details of syntax and semantics [9]. In a subsequent position paper, Cunningham et al. generated a list of the learning goals of conversational programmers based on a review of the literature, and proposed that an instructional approach that prioritizes explaining the high-level function, behavior, and structure of code would most closely align with these learning goals [10].

Despite the recent efforts to understand and support conversational programmers, we still lack essential information for informing a broader effort of learner-centered curriculum design for conversational programmers at the university level. Specifically, the extent to which students have conversational programming learning goals across various majors, industries, and demographics is unclear. As female students have a higher representation among non-majors in CS1 than CS majors [39], it is likely that conversational programmers do as well. Understanding conversational programmers' prevalence across different subpopulations of non-majors would inform the extent to which supporting conversational programming learning goals would broaden participation in computing. Additionally, current research has primarily focused on a few computing-adjacent majors, namely Information and Management Engineering, neglecting learners in non-STEM fields [6, 9]. This highlights the need for further exploration of conversational programmers across diverse educational contexts. Furthermore, existing research on potential learning goals of conversational programmers [10] relied on qualitative studies or small-scale surveys and has not explored the extent to which conversational programmers value these goals. Consequently, the results from these studies are challenging to generalize in larger or different contexts, leaving researchers unclear about conversational programmers' greatest needs as a group.

2.4 Expectancy-Value Theory

Eccles' Expectancy-Value Theory is a prominent theory in educational psychology that explains students' motivation and academic choices with a variety of factors [15]. Two core constructs directly impact motivation for a particular academic choice: expectancy of success and subjective task value. *Expectancy of success* is the students' belief in their likelihood of succeeding in performing a task. *Subjective task value* is students' value for the successful completion of the task. Students' *goals and self-schemata* have direct influence on both their expectancy of success of a task and their subjective task value. Students' *affective memories and reactions* to the task also influence subjective task values, as well as goals and self-schemata. Eccles et al. further proposed four components of subjective task value [17]: *attainment value* is the importance of performing the task well; *interest-enjoyment value* (also commonly termed *intrinsic value*) is the enjoyment students earn from their engagement with the task; *utility value* relates to the usefulness or instrumentality of the task to achieving students' goal; and *relative cost* is the negative outcomes arising from engaging with the task, like fear of negative consequences or the amount of effort [16, 47].

Figure 1: Overview of Expectancy-Value Theory components for our survey



Flake et al., moreover, formalized and proposed four dimensions to the costs of a task: *task effort* (the time, effort, or work required to engage in the task), *outside effort* (the time, effort, or work required for tasks unrelated to the primary interest), *loss of valued alternatives* (the sacrifices made to engage in the task of interest), and *emotional cost* (the psychological state resulting from exerting effort on the task) [18].

3 DATA AND METHODS

3.1 Survey Instrument

We designed a survey to investigate the prevalence and characteristics of undergraduate conversational programmers, as well as their perceptions and learning needs related to computing, based on Expectancy-Value Theory and prior research. We tested all our initial survey questions with two rounds of think-aloud sessions and a pilot survey with 52 participants. These steps were taken to inform the final version of the survey, ensuring students' understanding of the questions.

In the final version of our survey, we first asked questions about students' backgrounds, such as their gender, race, disability status[1] and the colleges they are enrolled in. Then, students were prompted to describe how they planned to engage with computing in their future jobs. A set of open-ended questions was asked to gather more in-depth insights into the challenges students faced and suggestions on how to improve the learning experience in non-majors computing courses for students like themselves. The rest of our survey questions were designed based on the constructs of Expectancy-Value Theory (see Figure 1), as follows:

- *Previous experience and associated affective memories*: We asked about the students' participation in pre-college experiences with programming as well as their associated emotions on a scale from 1 (Very negative) to 7 (Very positive).
- *Goals*: Questions related to conversational programmers' learning goals were designed based on Cunningham et al. [10] and were modified for clarity and conciseness based on findings from think-aloud sessions and a pilot survey.
- *Students' expectations for success*: Students' expectation for success was measured using Likert-type item questions. For example, students reported how good they feel they are at programming and at computing on a scale from 1 (Extremely bad) to 7 (Extremely good).
- *Subjective task value*: Based on Flake et al. [18], we asked seven-point Likert-type item questions related to four theoretical dimensions of perceived costs: loss of valued alternatives, task effort cost, emotional cost, and outside effort cost. Additionally, questions related to interest-enjoyment value and utility value were asked using seven-point Likert-type item questions, such as "To what extent do you agree with the following statement: I study computing because I enjoy it."
- *Achievement-related choice*: Students indicated the total number of computing courses they hope to take during their degree. Additionally, from a provided list of learning activities, they reported their level of interest in each activity.

For these questions, we mostly implemented seven-point Likert-type item questions, as seven-point scales are arguably the optimal number of choices for the reliability of survey questions [8]. We designed the survey so that the completion time would not exceed 15 minutes under usual circumstances.

[1]Specifically, respondents were asked whether they identify as having a disability or other chronic condition.

3.2 Participant Recruitment

Data were collected from students at a four-year, public higher education institution (the institution) during the 2023-2024 academic year. Our inclusion criteria were undergraduate students who have taken or are enrolled in at least one CS course and whose majors are not in CS, Computer Engineering, or a major that combines CS and another field. This study sought to recruit a representative sample of non-CS undergraduate majors at the institution. To do so, we requested a randomly selected sample of 5,000 students who met our inclusion criteria, constituting 44% of the eligible population, from our institution's data management office.

The computer science department at the institution offers several courses specifically designed for non-majors, including introductory programming in Python, introductory programming for engineers, a "big ideas of computing" course, and an introductory data science course. In addition, the for-majors CS1 has a majority of non-majors in its enrollment each semester, and non-majors regularly enroll in more advanced CS courses.

We distributed our anonymous, web-based survey via email to our sample. Two emails were sent: one initial invitation to take the survey and one reminder. We provided $5 Amazon gift card to incentivize students to complete the survey.

3.3 Analytic Sample

With a 15% response rate, our analytic sample consisted of 663 students. The sample was varied with regard to gender, ethnicity/race, disability status, and grade level. Specifically, 44.8% of the sample were women, and in terms of ethnicity/race, 48.1% self-identified as Asian/Asian American, 36.7% as White, 5.1% as Hispanic or Latinx, 1.5% as Black/African American, and 7.0% as other or multiracial. Additionally, 11.4% self-identified as having a disability or another chronic condition. Furthermore, 25.0% of the sample self-reported having CS minors, and the grade-level distribution was as follows: 32.7% freshmen, 35.6% sophomores, 27.3% juniors, and 4.4% seniors.

3.4 Data Analysis

We first classified the respondents into four desired computing endpoints based on their responses to our survey. The respondents were presented with definitions for the categories of non-programmers, conversational programmers, end-user programmers, and professional programmers, as established by Chilana et al. [6]. They were then asked to choose the profile that best describes their intended future involvement with computing. In our study, conversational programmers are students who selected "I will be speaking in the programmer's language (i.e., communicating about technical concepts to co-workers or clients) but I won't be writing code often." End-user programmers are those who selected "I will be writing programs or engaging in programming activities to support needs, goals, or work other than software development (e.g., data analysis for education, entertainment, accounting, or scientific research)."

Our analysis focused on conversational programmers to reveal their backgrounds, characteristics, and perceptions toward learning programming. We mainly used descriptive statistics (e.g., frequency distributions) to analyze conversational programmers' responses. To investigate the difference in the responses to two related questions within the same desired computing endpoint group (e.g. comparing programming and computing self-efficacy of conversational programmers), we relied on the Wilcoxon signed-rank test. This nonparametric test serves as an equivalent to the paired t-test when dealing with non-normally distributed data.

We further compared conversational programmers with end-user programmers to highlight any differences between these prominent groups of non-majors learning computing. To evaluate the statistical significance of differences in the distribution of responses to ordinal questions (e.g., Likert-type item questions) between conversational programmers and end-user programmers, such as the questions on subjective task value, we employed the Mann-Whitney U test (also known as Wilcoxon rank-sum test) considering its ordinal nature. For binary variables, we used two-sample tests of proportions (z-test) to assess the differences in proportions of conversational programmers and end-user programmers.

Lastly, we measured Spearman correlations among the components of Expectancy-Value Theory to explore relationships and gain insights into the motivational factors for enrolling in more computing courses and engaging with certain computing learning activities.

4 RESULTS

4.1 Desired computing endpoints among non-majors

First, we examined the desired computing endpoints of the respondents. Our results suggest that conversational programming is a commonly desired endpoint among non-majors. In our sample, the majority (40.9%) of students self-identified as conversational programmers, emerging as the largest group among non-majors, followed by end-user programmers (34.1%) (Table 1).

Table 1: Types of desired computing endpoints among non-majors

Non-programmer	**Conversational programmer**	End-user programmer	Professional programmer
143 (21.6%)	**271 (40.9%)**	226 (34.1%)	23 (3.5%)

4.2 Representation of Conversational Programmers

Representation across demographics. Next, we examined the prevalence of conversational programmers among various racial groups, genders, and disability status. Conversational programmers were not only the most common group among non-majors, but were the largest group within almost all demographics, except for men and non-binary respondents (Table 2). However, even for men and non-binary respondents, the shares of conversational programmers were the second-highest, closely following those of end-user programmers. Notably, large shares of conversational programmers were found among racial groups that are traditionally underrepresented in CS, such as Hispanic or Latinx and Black/African American. Moreover, about half of the students with disabilities self-identified

Table 2: Demographic characteristics, by types of desired computing endpoints among non-majors

Gender	N	Non-programmer	Conversational programmer	End-user programmer	Professional programmer
Man	345	15.4%	39.4%	40.6%	4.6%
Woman	297	29.0%	43.4%	25.6%	2.0%
Non-binary	14	21.4%	28.6%	50.0%	0.0%
Race/Ethnic					
Asian/Asian American	334	20.1%	38.9%	36.2%	4.8%
White	270	23.7%	42.2%	32.2%	1.9%
Hispanic or Latinx	54	22.2%	40.7%	35.2%	1.9%
Black/African American	17	17.6%	52.9%	17.6%	11.8%
Indigenous	5	0.0%	60.0%	20.0%	20.0%
Other or Multiracial	43	16.3%	48.8%	30.2%	4.7%
Disability					
Yes	74	18.9%	50.0%	29.7%	1.4%
No	576	22.0%	39.8%	34.4%	3.8%

Notes: The term 'Indigenous' includes American Indian/Alaska Native and Native Hawaiian/Pacific Islander.

Table 3: Colleges, by types of desired computing endpoints among non-majors

College	N	Non-programmer	Conversational programmer	End-user programmer	Professional programmer
Engineering	238	13.0%	39.5%	43.3%	4.2%
Liberal Arts & Sciences	186	24.2%	35.5%	34.9%	5.4%
Business	119	36.1%	45.4%	17.6%	0.8%
Information Sciences	35	5.7%	48.6%	40.0%	5.7%
General Studies	33	30.3%	33.3%	36.4%	0.0%
Fine & Applied Arts	24	8.3%	83.3%	8.3%	0.0%
Agricultural, Consumer & Environmental Sciences	18	27.8%	44.4%	27.8%	0.0%
Media	15	26.7%	26.7%	46.7%	0.0%
Applied Health Sciences	8	50.0%	12.5%	37.5%	0.0%
Education	6	16.7%	83.3%	0.0%	0.0%

Table 4: Majors, by types of desired computing endpoints among non-majors

College	Major	N	Non-programmer	Conversational programmer	End-user programmer	Professional programmer
Engineering	Mechanical Engineering	38	10.5%	55.3%	28.9%	5.3%
Engineering	Aerospace Engineering	31	6.5%	35.5%	54.8%	3.2%
Engineering	Civil Engineering	26	19.2%	53.8%	23.1%	3.8%
Liberal Arts & Sciences	Economics	37	10.8%	48.6%	40.5%	0.0%
Liberal Arts & Sciences	Statistics	25	12.0%	8.0%	76.0%	4.0%
Liberal Arts & Sciences	Mathematics	32	12.5%	25.0%	53.1%	9.4%
Business	Accounting	50	44.0%	44.0%	12.0%	0.0%
Business	Finance	41	36.6%	41.5%	22.0%	0.0%
Information Sciences	Information Science	34	5.9%	47.1%	41.2%	5.9%
General Studies	Undeclared	55	25.5%	47.3%	27.3%	0.0%

as conversational programmers, compared to slightly less than 40% of students without disabilities.

Representation across colleges and majors. We also explored conversational programmers' prevalence by colleges and majors. As shown in Table 3, conversational programmers were distributed across virtually all colleges, extending findings from earlier studies that confirmed the presence of conversational programmers in colleges such as Information. Additionally, conversational programmers were also heavily populated in colleges that were not previously studied, such as in Liberal Arts & Sciences (35.5%) and Business (39.5%).

Majors with 25 or more students are shown in Table 4. Mechanical Engineering, Civil Engineering, Economics, Accounting, Finance, Information Sciences, and undeclared majors were among those with over 40% of students self-identifying as conversational programmers.

4.3 Prior programming experience and associated affective memories

We inquired about conversational programmers' programming experiences prior to taking CS1, specifically, whether they learned programming during their K-12 education, via online courses, or

Figure 2: Prior programming experience

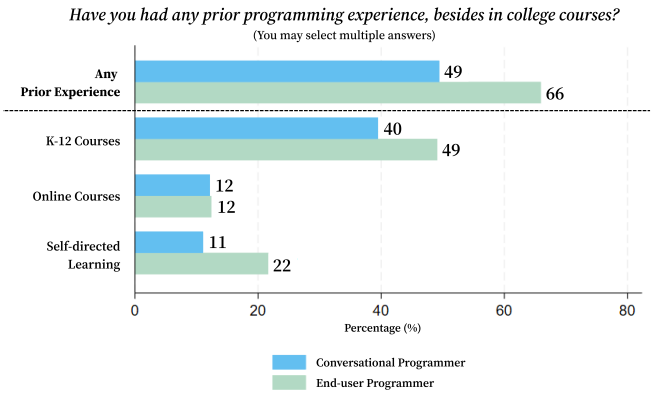
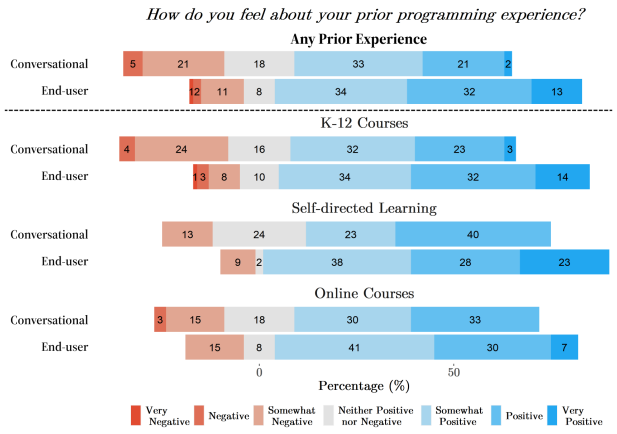


Figure 3: Sentiment toward prior programming experience



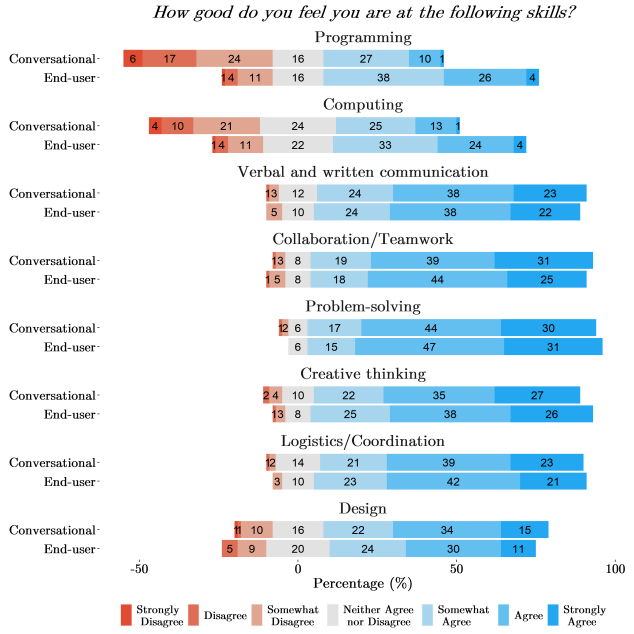
on their own. A little less than half of all conversational programmers in the survey (49.4%) had prior programming experience, and about 40% of all conversational programmers were exposed to programming via K-12 programming courses (see Figure 2). Slightly more than a quarter of conversational programmers responded that their prior experience with programming was at least somewhat negative, and 18.2% had a neutral response (see Figure 3). While most conversational programmers' prior programming experience was through K-12 education, their experience in K-12 courses was disproportionately negative compared to their experience in online courses or learning on their own (27.4% vs. 18.2% or 13.3%).

Comparison of conversational programmers and end-user programmers. In contrast, a larger share of end-user programmers than conversational programmers' had prior programming experience (49.4% vs. 65.9%, z=-3.70, p<0.01), and were exposed to programming during their K-12 education (49.1%) and on their own (21.7%) (Figure 2). End-user programmers generally had more positive prior experience than did conversational programmers (U=6640, p<0.01), and learning programming on their own was overwhelmingly positive for end-user programmers as nearly 90% of end-user programmers had at least a somewhat positive experience (Figure 3).

4.4 Expectation of success

We analyzed conversational programmers' expectation of success in computing[2] and in programming with two survey questions about their self-efficacy in these two categories. Our results suggest that many conversational programmers exhibited a low sense of self-efficacy in programming, which aligns with the findings from earlier studies [6, 9]. Specifically, 46.1% of conversational programmers felt that their programming skills are at least slightly bad, and only 10.7% regarded their programming skills to be at least moderately good (Figure 4). Interestingly, their sense of computing self-efficacy appeared to be higher. 31.4% of conversational programmers had greater self-efficacy in computing than programming while only 17.7% considered the reverse to be true (Wilcoxon z=-3.46, p<0.01).

Figure 4: Self-efficacy in programming, computing, and other skills



Unlike for programming or computing, conversational programmers had high self-efficacy in the other skills we asked about, which were chosen from skills we imagined would be important in conversational programmer's future work (see Figure 4). For more general skills like problem solving, collaboration/teamwork, communication, creative thinking, and logistics/coordination, at least 83% of conversational programmers thought that they were at least slightly good at each of skills, and no more than six percent of conversational programmers answered that they are at least bad. Conversational programmers had somewhat less self-efficacy for design, although more of these students felt more confident in their design skills than their computing or programming skills.

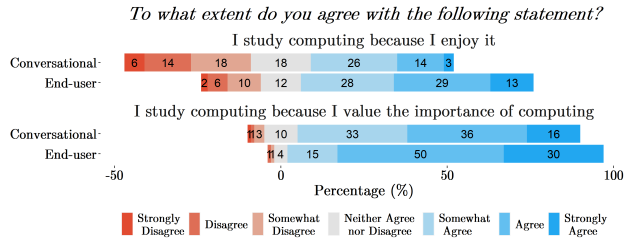
[2]We provided this definition of computing to survey respondents "Computing is a broader concept than programming and computer science; it's about all of the phenomena surrounding computing, including data, information, privacy, security, ethics, software engineering, and sociocultural and sociopolitical views of computing in society.", per [28].

Comparison of conversational programmers and end-user programmers. Both programming and computing self-efficacy among conversational programmers tended to be lower compared to that of end-user programmers (U=18152, p<0.01, and U=21509, p<0.01, respectively). Also, unlike conversational programmers, end-user programmers seemed to regard their programming skills to be better than their computing skills, although this difference was not significant (Wilcoxon z=1.64, p=0.10). Noteworthily, the Mann-Whitney U tests fail to reject that the distributions of responses of skills other than programming and computing differ significantly between conversational programmers and end-users.

4.5 Subjective task value

We further investigated conversational programmers' subjective task value for computing. Most conversational programmers were motivated to study computing because they valued the importance of computing – the utility value, and not necessarily because they enjoyed it or it was relevant to them – intrinsic value or attainment value. Notably, over 80% of conversational programmers studied computing due to its perceived importance, at least to some extent (see Figure 5). In contrast, less than half of conversational programmers (43.5%) studied computing because they at least somewhat agreed with the statement that they enjoyed learning computing. Further, the share of conversational programmers who studied computing while not enjoying it (37.6%) was approximately eight times higher than the share of them who studied computing while not recognizing its utility value (4.8%).

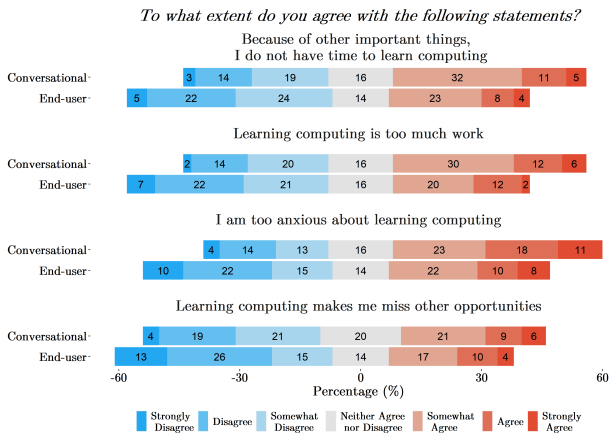
Figure 5: Interest-enjoyment and utility value



We then analyzed conversational programmers' relative or perceived costs that may decrease their value for computing, and demotivate their further pursuit of computing. Among the four types of costs proposed by Flake et al.[18], emotional cost was considered as the top concern for most conversational programmers. Over 50% of conversational programmers at least somewhat agreed that they felt too anxious about learning computing (see Figure 6). Additionally, slightly less than 50% conversational programmers at least somewhat agreed that they do not have time to study computing or that learning computing is too much work, suggesting they may be hindered by "loss of valued alternatives" or "task effort costs", as per Flake et al..

Comparison of conversational programmers and end-user programmers. End-user programmers tended to agree more with the prompts asking whether they study computing because they enjoy it or because they value its importance than conversational programmers (U=19783.5, p<0.01 and U=21101, p<0.01). Furthermore, anxiety

Figure 6: Relative costs of studying computing

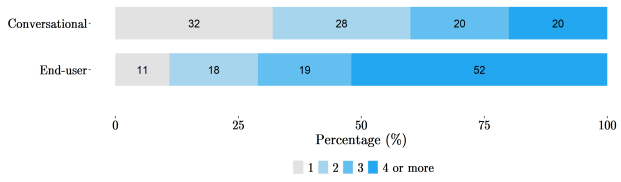


was the most important cost preventing both conversational programmers and end-user programmers. For both conversational programmers and end-user programmers, moreover, perceptions that not having the time to learn computing and learning computing being too much work were not significantly different (Wilcoxon z=-0.13, p=0.90 and Wilcoxon z=0.20, p=0.84). Nevertheless, conversational programmers perceived all four categories of costs a greater hindrance to studying computing than end-user programmers did (U=25167, p<0.01; U=24812, p<0.01; U=24645, p<0.01; and U=26248, p<0.01).

4.6 Computing course needs

We asked respondents how many computing courses, including their current or past enrollment, they intend to take during working toward earning their degrees. As shown in Figure 7, 68% of conversational programmers were motivated to take at least one additional computing course after their first computing course, with 20% of them wanting to take four or more computing courses during their degree.

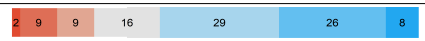
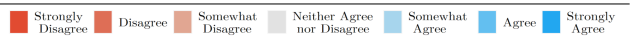
Figure 7: Intended number of computing courses



Comparison of conversational programmers and end-user programmers. However, when comparing conversational programmers with end-user programmers, conversational programmers, on average[3], preferred to take fewer computing courses: while 60% of conversational programmers intended to take at most 2 computing courses, over 50% of end-users stated that they wanted to take at least four or more computing courses. The distribution of additional course needs were significantly different (U=18115, p<0.001).

[3]This average was calculated based on the assumption that students indicating a desire to take four or more courses had, on average, a similar number of courses in mind.

Table 5: The extent to which conversational programmers agree with the hypothesized learning goals

Goal Types (Clusters From [10])	Conversational Programmers' Learning Goals	
Recognize computational and workplace processes	1. Understand the work of technical teams	
	2. Understand customer needs related to their use of technologies	
	3. Have general knowledge about concepts related to programming and computer science	
Build and modify with code	4. Perform data analysis or improve efficiency by using computing	
	5. Build functional prototypes	
	6. Modify code (e.g., fix small issues or make updates)	
Measure code's ability to meet goals	7. Evaluate whether the technical product is successfully implemented (e.g., find bugs; compare actual and designed functionality to assess product standards)	
Articulate goals to be achieved with code	8. Specify goals for creation to programmers (i.e., describe what developers needs to do in language that developers can understand)	
	9. Use a mockup to demonstrate goals (e.g., design a website using graphics tools or create a demo prototype to articulate ideas to developers)	
Find relevant technical information	10. Gain new technical information on my own, either by asking a technical expert or reading resources	
Predict what is possible to achieve with code	11. Understand applications and benefits of technologies and programming related concepts	
	12. Understand whether a new idea is technically feasible (i.e. know what programmers can and cannot do)	
Describe code functionality	13. Have big-picture conversations about how computing technologies work with non-technical audiences	
	14. Communicate with developers about code	
	15. Describe which parts of the technologies or code are responsible for achieving certain business/customer priorities (i.e., connect technical processes with business priorities)	
Establish strong working relationships with developers	16. Be respected and considered credible by people with technical backgrounds	
Have skills valuable to the job market	17. Have skills valuable to the job market, such as familiarity with recent technologies and marketable programming languages	
		

4.7 Computing learning goals

Consistent with previous studies on how conversational programmers learn computing for the reasons other than writing programs [6, 45], their learning goals do not appear to be focused on programming at a professional level. Indeed, fewer than half of conversational programmers either somewhat agreed, agreed, or strongly agreed that they would want to be able to write code at a professional level.

To understand the learning goals of conversational programmers, Cunningham et al. proposed a set of hypothesized learning goals for this group, based on a review of prior research [10]. In our survey, we evaluated respondents' level of agreement with these goals (see Table 5). Notably, these conversational programmers' learning goals were more valued by conversational programmers than the traditional goal of writing code at a professional level, with over 60% at least somewhat agreeing with all of the hypothesized conversational programming goals.

However, the extent of agreement varied across the different learning goals. Aligned with prior studies, most (87.5%) conversational programmers aspired to learn skills that were valuable to the job market, such as familiarity with recent technologies and marketable programming languages. Further, more than 80% of conversational programmers desired to be respected and considered credible by people with technical backgrounds. Yet, their interests in describing code functionality seemed to be lower. Specifically, just below 70% of conversational programmers wanted to be able to communicate with developers about code, while slightly more than 70% aimed to be able to have big-picture conversations about how computing technologies work with non-technical audiences. Moreover, most conversational programmers appeared to place importance on recognizing computational and workplace processes. For example, they wanted to have general knowledge about concepts related to programming and CS (85.2%) and understand customer needs related to their use of technologies (81.9%).

The results further revealed that conversational programmers are less interested in goals that are associated with building and modifying code, measuring code's ability, and articulating goals of code. The share of conversational programmers who at least somewhat agreed that these learning goals are important to them were consistently a little over 60% while the average for other learning goals were 77.7%.

Popularity of conversational programmers' learning goals among non-majors. Table 6 presents both a focus of the traditional undergraduate CS curriculum and learning goals proposed for conversational programmers [10] as well as the proportion of learners of different desired computing endpoints who at least slightly agree with each item, indicating its importance to them. Strikingly, learners with different desired computing endpoints also showed a greater interest in conversational programmers' learning goals compared to the focus of the traditional undergraduate CS curriculum of writing code at a professional level. For instance, non-programmers in the sample were approximately three to ten times more likely to rank conversational programmers' learning goals higher than the goal of writing code at a professional level, compared to the opposite. Both end-user and professional programmers also seemed to value being able to achieve conversational programmers' learning goals. In particular, end-user programmers seemed to place higher value on certain conversational programmers' learning goals, such as predicting what is possible to achieve with code (learning goals 11 and 12), compared to the skill of writing code at a professional level. Notably, for end-user programmers, the ability to articulate what can be achieved with code (learning goals 8 and 9) was the only area significantly less valued than the traditional goal of coding proficiency (Wilcoxon z=2.49, p=0.01; Wilcoxon z=4.55, p<0.001). Professional programmers also appeared to be interested in these conversational programmers' learning goals, valuing them as much as professional coding skills, with the exception of using a mockup

Table 6: Preference for conversational programmer versus a traditional learning goal across different desired endpoints

Focus of the Traditional Undergraduate Computer Science Curriculum	Non-	Conversational	End-user	Professional
Write code at a professional level	29%	45%	77%	87%

Conversational Programmers' Learning Goals (Adapted from [10])	Non-	Conversational	End-user	Professional
1. Understand the work of technical teams	57%	78%	81%	87%
2. Understand customer needs related to their use of technologies	72%	82%	79%	87%
3. Have general knowledge about concepts related to programming and computer science	71%	85%	88%	96%
4. Perform data analysis or improve efficiency by using computing	71%	75%	89%	91%
5. Build functional prototypes	48%	63%	78%	83%
6. Modify code	52%	63%	85%	91%
7. Evaluate whether the technical product is successfully implemented	48%	63%	76%	74%
8. Specify goals for creation to programmers	46%	65%	66%	91%
9. Use a mockup to demonstrate goals	41%	62%	65%	74%
10. Gain new technical information on my own, either by asking a technical expert or reading resources	57%	76%	87%	96%
11. Understand applications and benefits of technologies and programming-related concepts	66%	77%	89%	91%
12. Understand whether a new idea is technically feasible	60%	79%	86%	91%
13. Have big-picture conversations about how computing technologies work with non-technical audiences	57%	73%	79%	74%
14. Communicate with developers about code	45%	68%	81%	91%
15. Describe which parts of the technologies or code are responsible for achieving certain business/customer priorities	54%	69%	71%	91%
16. Be respected and considered credible by people with technical backgrounds	66%	81%	84%	87%
17. Have skills valuable to the job market, such as familiarity with recent technologies and marketable programming languages	76%	87%	89%	91%

to demonstrate goals and engaging in high-level discussions about computing technologies with non-technical audiences.

4.8 Learning activities

Lastly, we measured conversational programmers' interests toward various learning activities. Conversational programmers generally showed an interest in understanding code with guidance and expressed a preference for interactive learning activities (see Figure 8). For instance, they were interested in interactive code review sessions, lectures with live coding, and writing code with guiding comments. In contrast, they seemed to be less interested in writing code from scratch. Surprisingly, relatively few conversational programmers showed interest in learning activities centered on communicating about programming, except for presenting how programming can be applied to a problem. For example, approximately 30% of conversational programmers indicated that they were at least slightly disinterested in learning activities such as listening to a technical conversation about a program to identify relevant code segments or discussing technical concepts in plain language.

Comparison of conversational programmers and end-user programmers. As displayed in Figure 8, end-user programmers exhibited similar patterns of preferences toward learning activities as did conversational programmers, but greater. Besides making presentations (U=28372, p=0.15), all learning activities were more preferred by end-user programmers than by conversational programmers. These results are in line with the ones in section 4.7.

4.9 Correlation between factors associated with motivation

Theoretically, each component of Expectancy-Value Theory is correlated [17]. Thus, we examined the correlations among these components within the framework of Expectancy-Value Theory, specifically in relation to the motivation of conversational programmers to enroll in computing courses and engage in certain learning activities (see Figure 1).

Factors directly associated with intended number of computing courses. Eccles' Expectancy-Value Theory suggests that learners' motivation for achievement-related choices, such as enrolling in computing courses, is directly guided by their expectation of success

Figure 8: Conversational programmers' interests in various learning activities

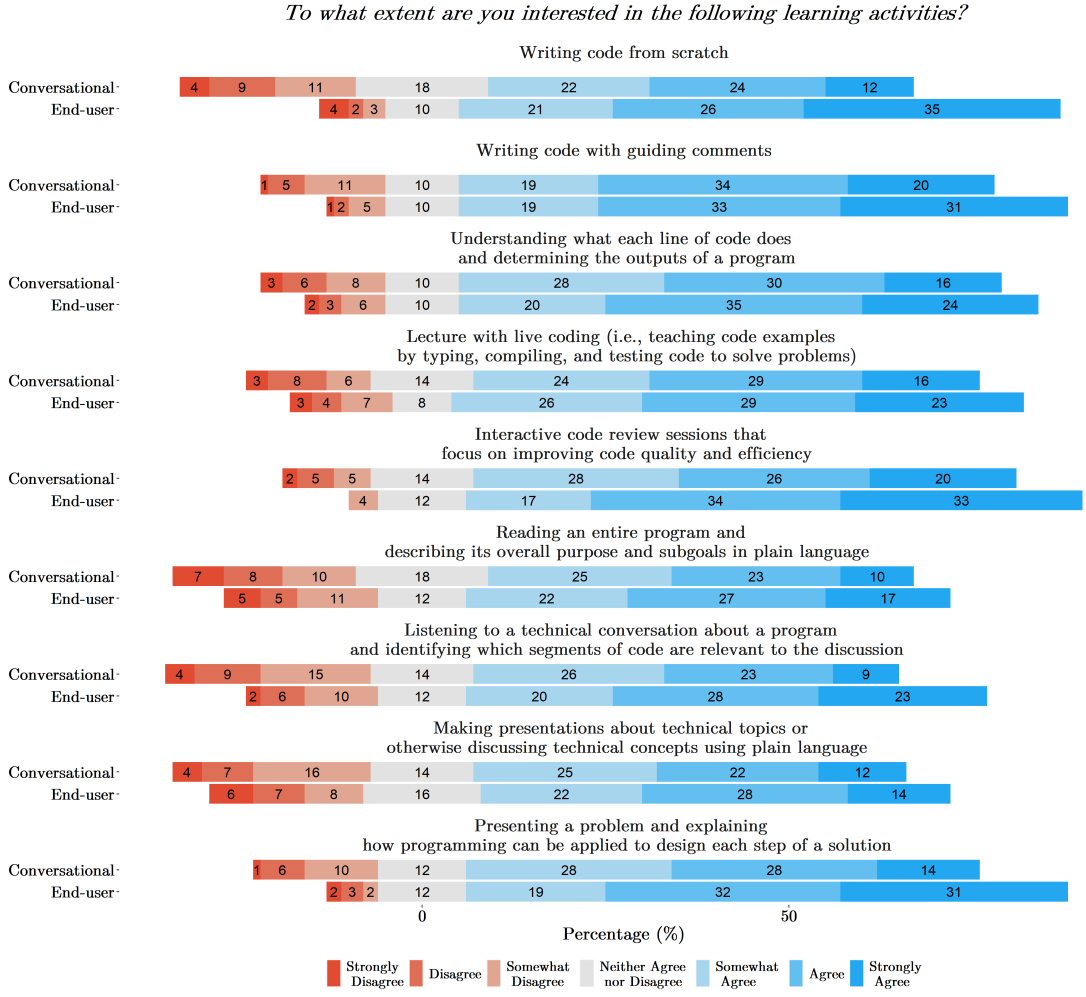
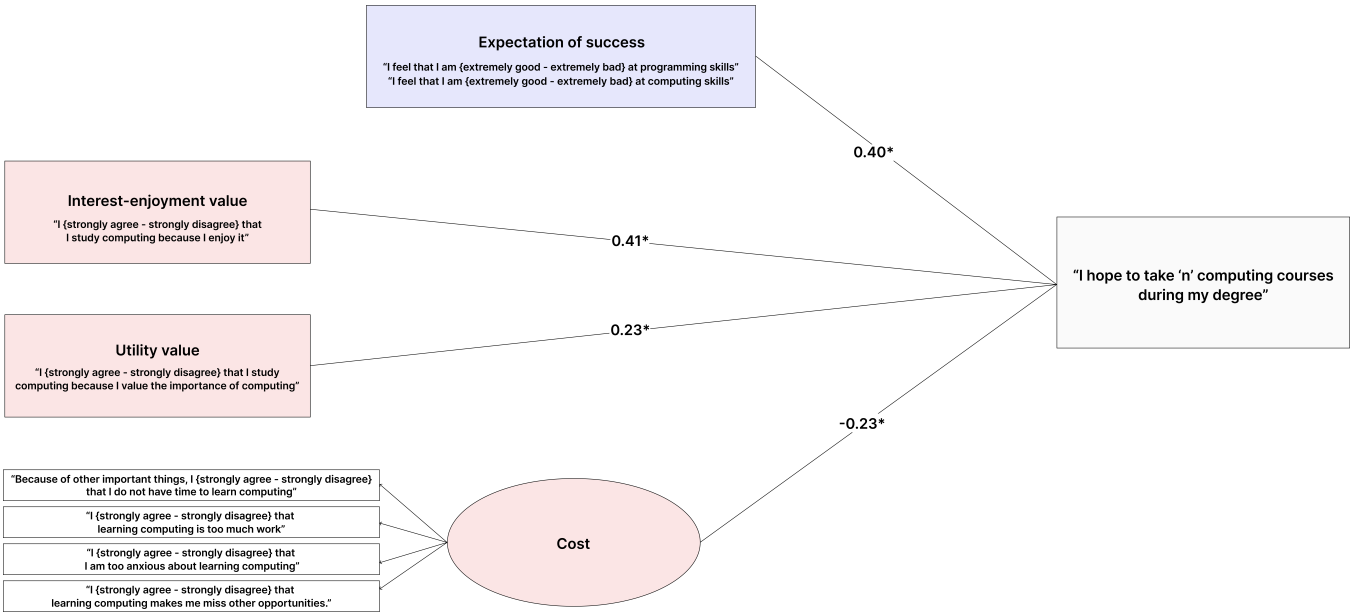


Figure 9: Correlation analysis of conversational programmers' motivational factors for enrolling in more computing courses



and subjective task value. Therefore, we examined the correlations between conversational programmers' intended number of computing courses, self-efficacy in computing and programming, and subjective task value Figure 9 and confirmed that these relationships hold.

Our analysis revealed a positive monotonic correlation between the number of computing courses conversational programmers intend to take during their degree and their expectation of success (r_s = 0.40, p < 0.001) . Furthermore, the enjoyment that conversational programmers derive from computing positively correlated with the number of computing courses they planned to take (r_s = 0.41, p < 0.001). The value conversational programmers placed on the importance of computing was positively but relatively weakly correlated (r_s = 0.23, p < 0.001).

In contrast, their perceived cost was negatively correlated with their planned course load (r_s = -0.23, p < 0.001). Specifically, viewing computing as overly burdensome showed a stronger negative correlation with the planned course load (r_s = -0.22, p < 0.001), while thinking they do not have time to learn computing due to other important things exhibited a weaker negative correlation with the intended number of computing courses (r_s = -0.14, p < 0.01).

Factors directly associated with interests in computing learning activities. We also examined the correlations between conversational programmers' interests in engaging in different kinds of computing activities, expectation of success, and subjective task value (see Figure 10). Conversational programmers' interest in learning activities that emphasize conversation (learning activities 6, 7, 9), alongside activities that focus on code (learning activities 1, 2, 3) showed a positive correlation with their expectation of success ($0.14 \leq r_s \leq 0.32$). Similarly, some of the learning activities centered on conversation (learning activities 7, 9) and writing code from scratch (learning activity 1) exhibited a weak negative correlation with perceived cost ($-0.19 \leq r_s \leq -0.14$), implying that conversational programmers with high perceived cost may be less interested in these activities.

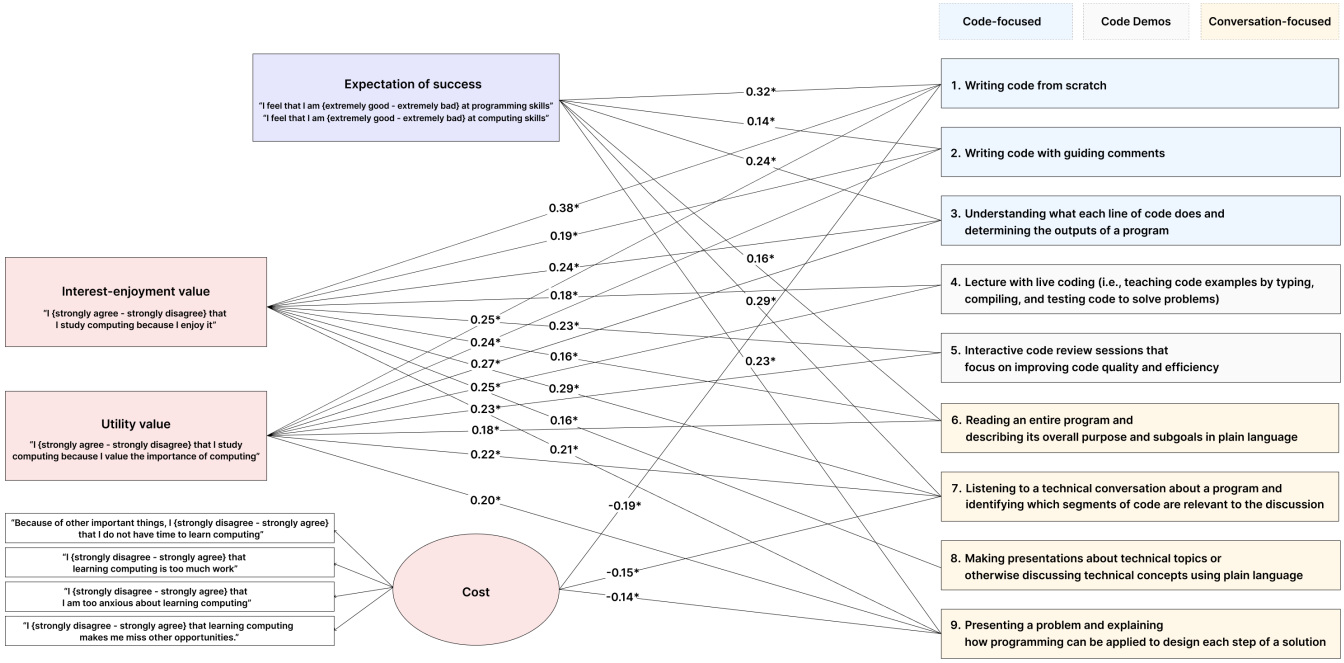
Moreover, expectation of success was not significantly correlated with interest in lectures with live coding (learning activity 4) or interactive code review sessions (learning activity 5), and the correlation with interest in writing code with guiding comments (learning activity 2) was weak. Interestingly, among all listed learning activities, only these three showed significantly positive correlations with the top three skills of conversational programmers: problem-solving, collaboration/teamwork, and communication.

Additionally, we found that each learning activity positively correlated with the interest-enjoyment value ($0.16 \leq r_s \leq 0.38$), suggesting that greater interest in these activities was associated with the increased enjoyment or intrinsic motivation for learning computing. Interestingly, learning activity 8 was the only activity among all those listed that did not correlate with utility value (r_s = 0.08, p = 0.17).

Expectation of success and subjective task value. All subjective task values were statistically correlated with expectation of success, as expected from the theory. Expectation of success showed a strong correlation with interest-enjoyment value (r_s = 0.47, p < 0.001) and a relatively weak correlation with utility value (r_s = 0.22, p < 0.001). In contrast, perceived cost was negatively correlated with expectation of success (r_s = -0.34, p < 0.001).

Factors associated with computing learning goals. The theory suggests that individuals' goals directly influence their expectation of success and subjective task value. Thus, we examined correlations between conversational programmers' clusters of learning goals

Figure 10: Correlation analysis of conversational programmers' motivational factors for engaging with certain learning activities



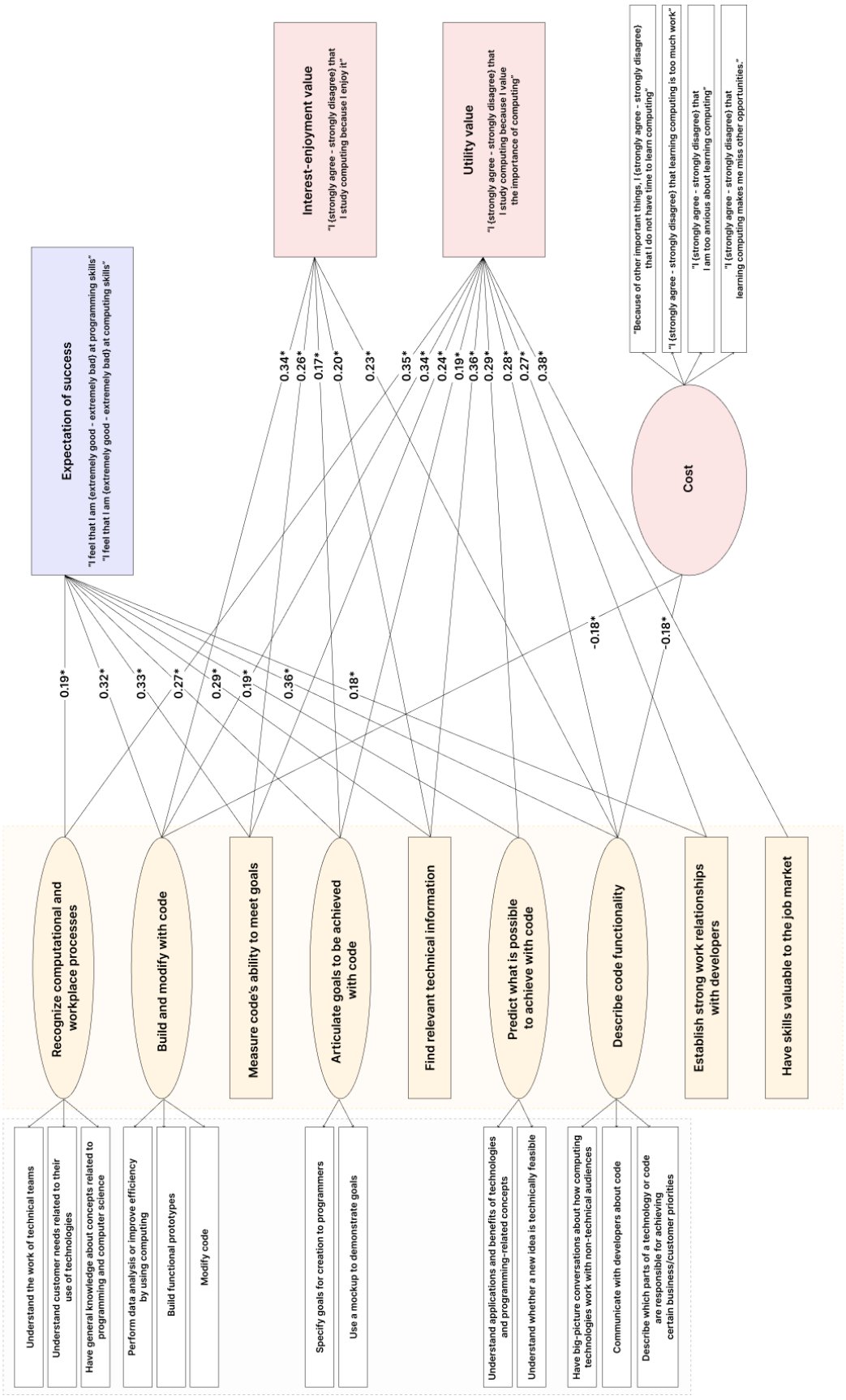
(or goal types[4]) and these elements, and confirmed these relationships in most cases (see Figure 11). In terms of the correlations between conversational programmers' goal types and their expectation of success[5], all goal types showed significant correlations with expectation of success, except for the goal type related to having skills valuable to the job market. Notably, goal types related to the ability to describe code's functionality (r_s = 0.36, p < 0.001), measure code's ability to meet goals (r_s = 0.33, p < 0.001), and build and modify with code (r_s = 0.32, p < 0.001) demonstrated relatively strong correlations with expectation of success. Interestingly, these goal types exhibited correlations with expectation of success that were comparable to, or even exceeded, those associated with the traditional goal of achieving proficiency in coding ($r_s = 0.32, p < 0.001$). Furthermore, these three goal types were also relatively strongly correlated with interest-enjoyment value ($r_s = 0.34, p < 0.001$, $r_s = 0.26, p < 0.001$, and $r_s = 0.23, p < 0.001$ respectively), while other goals were either weakly correlated or not significantly correlated with interest-enjoyment value. Unlike the correlation between learning goals and interest-enjoyment value, all learning goals were significantly positively correlated with utility value. We also investigated the correlation between learning goals and perceived cost[6]. Only two learning goals, building and modifying with code ($r_s = -0.18, p < 0.01$) and describing code functionality ($r_s = -0.18, p < 0.01$), showed negative correlations with perceived cost.

We further explored the correlation between conversational programmers' learning goals and achievement-related choices, as they can be indirectly associated according to Expectancy-Value Theory. While all goal types showed positive correlations ($0.15 \leq r_s \leq 0.31$), describing code functionality demonstrated the strongest positive correlation (r_s = 0.31, p < 0.001) with intended number of courses. Furthermore, describing code functionality was relatively strongly correlated with conversation-focused activities ($0.27 \leq r_s \leq 0.37$, p < 0.001).

Factors associated with affective memories. Expectancy-Value Theory suggests that affective memories about previous achievement-related experiences affect both learners' goals and subjective task value, although these relationships were not supported in every case. For example, conversational programmers' affective memories of previous programming experiences showed a significant correlation with only some of the conversational learning goals: building and modifying with code ($r_s = 0.26, p < 0.01$), measuring code's ability to meet goals ($r_s = 0.24, p < 0.01$), and describing code functionality ($r_s = 0.19, p < 0.03$). On the other hand, affective memories were significantly correlated with subjective task value, as predicted by the theory. Although all associations were significant, affective memories showed a stronger positive correlation with interest-enjoyment value than with utility value (r_s = 0.49, p < 0.001 vs. r_s = 0.22, p = 0.01). Moreover, affective memories were negatively correlated with perceived cost (r_s = -0.21, p < 0.01).

[4]Goal types were derived from the average Likert scale responses to relevant learning goals (refer to Figure 11).
[5]Expectation of success was calculated as the mean of programming and computing self-efficacy.
[6]Perceived cost was calculated by averaging four sub-components of relative cost: task effort, outside effort, loss of valued alternatives, and emotional cost.

Figure 11: Correlation analysis of conversational programmers' learning goals, expectation of success, and subjective task value



5 DISCUSSION

Our study provides an overview of the learning needs of undergraduate conversational programmers, in addition to their demographics and characteristics. Our study replicates many prior findings about conversational programmers at the university level, highlights opportunities for novel curricular design, and shows that a focus on conversational programmers' goals and challenges could be instrumental in broadening participation in computing among non-majors.

5.1 We validate previous findings across a broader sample of undergraduates

Our study does not find any major surprises in the desired learning goals or characteristics of undergraduate conversational programmers. We validate conversational programmers' interests in a previously proposed set of learning goals of conversational programmers [10], showing that these types of computing knowledge are valued by this population much more than code writing. We also replicated findings about conversational programmers' affective characteristics. Specifically, we show that conversational programmers have low self-efficacy in computing and even lower self-efficacy in programming. We verify the claim that conversational programmers have less time to learn computing, as they prefer to take fewer classes and a large proportion agree that they "do not have time to learn computing." We further extend the literature's understanding of conversational programmers by finding that the anxiety and workload are other factors that may similarly contribute to the costs of learning computing.

5.1.1 Conversational programmers are different than end-user programmers. Prior work differentiated conversational programmers from end-user programmers based on their desired goals [6]. We show that these two groups differ in other characteristics as well. In all metrics that predict motivation for programming and computing, according to Eccles' Expectancy-Value Theory [15], we find that conversational programmers have responses that predict less motivation, compared to their end-user programmer peers. Regardless of their higher self-efficacy in computing than in programming, conversational programmers' expectation of success in both computing and programming is lower than end-user programmers'. Their sense of the costs of learning computing is higher than that of end-user programmers, whereas both their interest-enjoyment value and their utility value are lower. In addition, their memories of prior computing experience are more negative. It is no wonder that conversational programmers intend to take fewer computing courses than end-user programmers, and also have less interest in computing learning goals.

5.2 Conversational programmers are a suitable target for broadening participation

For multiple reasons, our findings suggest that supporting conversational programmers' learning needs can broaden participation in computing.

5.2.1 High representation among non-majors. This study presents evidence that, among non-majors taking computing courses, individuals historically underrepresented in CS, namely women, Hispanic/Latinx and Black/African American individuals, and students with disabilities, are more likely to describe themselves as conversational programmers than end-user programmers or professional programmers. In other words, better supporting conversational programmers may empower underrepresented individuals to participate in CS, which would subsequently contribute to widening pathways to computing.

Further, conversational programmers are well-represented across non-majors hailing from many parts of campus. The conversational programming endpoint is a goal that non-majors from Engineering to Business to Liberal Arts all share, suggesting that conversational programmer learning goals could be appropriate for a general education course accessible to multiple majors.

5.2.2 Conversational programmers have overlapping learning goals with other endpoints. We not only validated that the conversational learning goals proposed in prior research are valuable to conversational programmers, but also demonstrated that, for the most part, these goals are shared by non-majors with end-user programming and professional programming endpoints. It follows that focusing on these learning outcomes earlier in computing pathways may not only benefit conversational programmers, but also the majority of non-majors, to meet their desired goals.

Together, conversational programmers and end-user programmers made up the lion's share of the non-majors we profiled. While these groups differed in most items we measured, it is worth noting that the boundary between the end-user programmer endpoint and the conversational programmer endpoint is not strict. Rather, it seems that there is a spectrum of desired engagement with programming, as we see variations within both groups of students as far as how much programming they value. While it was not the most popular learning goal, many conversational programmers expressed interest in being able to modify code. A minority even showed interest in writing code at a professional level. Conversely, the majority of end-user programmers wanted to be able to communicate about computing with non-technical audiences.

5.3 Towards learner-centered curriculum design for conversational programmers

With the reasons for supporting conversational programmers established, the next question is how to best support this population with undergraduate computing courses that meet their needs. We review undergraduate conversational programmers' key challenges and suggest ways to address them via curriculum design.

5.3.1 Challenges of motivating this population. Broadening participation in computing arguably starts with providing students a positive experience in their first computing course. Our findings suggest that conversational programmers, in particular, would face several challenges in conventional computing courses. First, a larger share of conversational programmers reported that they had negative prior programming experience, which is significantly correlated with interest-enjoyment value in computing. Indeed, conversational programmers had low interest-enjoyment value in programming

compared to their other skills. Moreover, coding-intensive learning goals and activities were relatively strongly correlated with expectation of success, which may have contributed to why conversational programmers displayed little interest in them.

An additional challenge to consider with respect to curriculum design is that conversational programmers have much lower interest-enjoyment value toward programming/computing than utility value, and interest-enjoyment value is more highly correlated with the number of additional courses they intended to take. Moreover, conversational programmers often report lower self-efficacy in programming and computing, which is also associated with their motivation to take more computing courses. To encourage the participation of conversational programmers in computing, curriculum design should primarily consider raising their interests in computing and bolstering their expectation of success. One possible approach is to design a learner-centered curriculum for a first computing course that does not focus on coding-intensive activities. Instead, it could align its content and activities with the learning goals conversational programmers aim to achieve and may be less affected by their low interest-enjoyment value or expectation of success. These goals include recognizing computational and workplace processes, predicting what is possible to achieve with code, and establishing strong working relationships with developers.

In addition, emotional cost is one of the greatest deterrents for conversational programmers. While we cannot change conversational programmers' major requirements or other commitments, we may be able to reduce their anxiety associated with computing courses by reducing the concurrent workload. This is especially important considering how viewing computing as overly burdensome was negatively correlated with their planned course load. We recommend designing multiple computing courses for non-majors as shorter unit courses (e.g., 1-unit courses).

5.3.2 Unexpected results. Surprisingly, our survey results revealed that learning goals related to code communication are not prioritized by conversational programmers. Relatedly, they were less interested in conversational learning activities. We suspect that the reasons conversational programmers are reluctant to engage in communication activities – which are largely public and involve explaining domain expertise – may be attributed to their low self-efficacy and high emotional cost (i.e. anxiety about computing). Describing the overall purpose of a piece of code or presenting about how programming can solve a problem not only require a relatively advanced understanding of computing, but also require showing that understanding to others. Supporting our suspicion, we observed a relatively stronger positive correlation between these learning goals and activities with expectations of success, and a more pronounced negative correlation with costs, compared to other learning objectives and activities. Learning goals related to describing code functionality were also most strongly correlated with the intended number of courses, implying that conversational programmers who plan to take more computing courses are more interested in code-focused goals.

Another unexpected result was that a surprising number of conversational programmers wanted to take three or more computing courses. While most studies on undergraduate conversational programmers focus on CS1, our findings suggest the importance of understanding the needs and challenges facing conversational programmers across multiple courses.

5.3.3 Opportunities to meet their needs. In our review of our results, we see signals suggesting how to design learning activities for conversational programmers. We suggest a two-stage process: First, introductory computing course(s) that focus on increasing their expectation of success and interest-enjoyment value, and subsequently, more advanced course(s) that concentrate on conversational goals.

In the first stage, we recommend learning activities that play to the strengths of conversational programmers in order to build self-efficacy and interests. Conversational programmers have high self-efficacy in other, non-computing areas such as collaboration, communication, coordination, and problem-solving, all of which seem like natural fits for activities that support the conversational programming endpoint. Building curricula that leverage these competencies, like interactive code review sessions, could be particularly engaging and effective.

Moreover, based on the evidence from our study and prior work, we argue that conversational programmers are likely to respond positively toward learning activities that are scaffolded, such as writing code with guiding comments, irrespective of their expectations of success and perceived costs. Therefore, it may be necessary to utilize, or design, scaffolded activities appropriate to conversational programmers' goals. This approach mirrors the efforts so far to support conversational programmers, where certain types of scaffolded activities appear to increase self-efficacy and interest-enjoyment value among conversational programmers [9].

Additionally, we recommend tailoring learning goals and activities to match the utility values of conversational programmers. A lot of conversational programmers study computing because because they value the importance of it. Thus, explicitly setting the learning goals to be employment-oriented could be an effective approach to increase motivation, given employment-oriented goals' strong correlations with utility value, as opposed to interest value. A notable factor contributing to conversational programmers' sense of failure is the mismatch between their expectations and the alignment of computing courses with their learning objectives [45]. A critical consideration is identifying activities that conversational programmers perceive as useful and authentic. Although we initially hypothesized that activities such as presenting technical topics or discussing technical concepts in plain language would be highly relevant, conversational programmers may find them less valuable, as implied by their weak correlation with utility value.

By aligning learning goals with conversational programmers' needs, enhancing their expectation of success and interest-enjoyment, and gradually introducing more specialized goals and activities that are related to conversations, we may be able to effectively support conversational programmers in achieving their desired endpoint.

6 THREATS TO VALIDITY

While offering insights of non-majors taking CS courses, particularly of conversational programmers at a broad scale, our study nevertheless poses several limitations. First, even though we reached out to a random sample, the 15% response rate renders our survey vulnerable to unobserved confounding. Studies have shown that

response rates are higher among female respondents [41], as well as respondents' who have had positive experience regarding the survey matter [37]. We did see these two influences in our analytic sample. Our respondents were more likely to be women[7] than the target population (46.3% vs. 36.3%), and they were also more likely to have a CS minor than the population (25.0% vs 8.2%). In addition, our respondents were more likely to be Asian than the population (48.1% vs 24.5%), and less likely to be Black (2.6.% vs 3.5%) or Hispanic/Latinx compared (8.1% vs 11.0%) to our target population[8]. Perhaps because our survey email mentioned "non-major computing courses" in the title, we had a stronger response from students who have a more positive attitude towards computing and interest in taking a larger number of computing courses, including CS minors. If this is true, it follows that undergraduates with less positive experiences with programming, those with non-programming goals, and those who plan to take fewer computing courses are likely underrepresented in our analytic sample.

Another limitation is that we have disproportionate sample sizes for students of some demographic groups, in some colleges, and in class standing. For example, the sample size for Black/African American conversational programmer was nine students (Table 2), and that of students in the College of Education was only six students (Table 3). These sample sizes would generally be considered to be too small to draw a credible conclusion about these groups. Also, the response rate of senior students was unusually small compared to their density in the population (4.4% vs. 44.78%). In addition, we report results from only one institution. While computing is popular across many colleges and universities [3], it is possible that some characteristics of our institution are associated with non-majors having an interest in computing. These may bias the population representation of our sample and consequently our findings.

To reduce the survey fatigue of respondents and increase the response rate while maintaining comprehensiveness of the survey, our survey contained single item questions instead of multi-item. Although some studies suggest that single-item questions present similar level of validity on their constructs compared to multi-item questions [2, 48], relevant literature commonly utilizes multi-item questionnaires. Our choice with survey design further prevents us from employing common survey analysis methods, such as factor analysis, which estimates the validity of survey constructs [46].

Moreover, our study conducted multiple statistical tests, for which some researchers employ multiple testing corrections, such as the Bonferroni correction, to account for potentially inflated Type I errors [1, 43]. However, there is some controversy over whether or not to use these corrections, especially considering that they can increase Type II errors, potentially missing significant findings [43]. Indeed, the circumstances of the study are important in determining whether or not to use these corrections [1]. For our study, we did not apply these methods, as our hypotheses are grounded in the existing literature and Type II errors are more critical in our setting.

[7]While our survey asked respondents to indicate their gender (*i.e.* whether they consider themselves as women, men, or non-binary), we were provided with the number of female students in the population that meet our inclusion criteria.

[8]The classification of race might slightly differ from that of the data management office of the institution.

Lastly, the quantitative nature of the analysis did not provide us further insights as to *why* the participants answered the questions as did they, which could be supplemented with a qualitative analysis of some of the respondents. Future research could delve deeper into the motivation of conversational programmers for learning more computing and engaging in certain computing activities through in-depth qualitative studies.

7 CONCLUSION

Despite the influx of non-CS major students into computing courses, an understanding of these learners' varied goals and attitudes has been limited. Particularly, conversational programmers, whose primary learning goals are to understand computing rather than to program, have arguably been incidental in both classrooms and computing education research. In this study, we found that students who self-identify as conversational programmers are highly represented among the non-major population, as well as among subpopulations underrepresented in computing. We also found that this population faces increased barriers compared to other types of non-majors, such as their discrepancy in intrinsic and utility value toward, relatively negative past experience in, and anxiety about computing. Developing a learner-centered curriculum that supports the specific needs of this broad and diverse population may be an avenue for broadening participation in computing. Such an instructional approach may require re-thinking the content and focus of introductory computing courses to better bolster their self-efficacy and interest while aligning with their desired goals. Our results suggest that such a change can not only meet the desired learning goals of conversational programmers, but also other groups of non-majors, like end-user programmers.

ACKNOWLEDGMENTS

We thank our lab members Arif Demirtaş and Yoshee Jain for their support throughout the project. We would also like to express our gratitude to our colleagues, Colleen Lewis, Morgan Fong, Andrea Watkins, Victor Zhao, Max Fowler, and Andrew Chen for their valuable feedback and insights. We extend our appreciation to Lu Ting from the University of Illinois Division of Management Information and Michael Kang for their support in the facilitation of this project. This project is funded by the Department of Computer Science at the University of Illinois Urbana-Champaign.

REFERENCES

[1] Richard A. Armstrong. 2014. When to use the Bonferroni correction. *Ophthalmic and Physiological Optics* 34, 5 (2014), 502–508.

[2] Lars Bergkvist and John R. Rossiter. 2007. The Predictive Validity of Multiple-Item versus Single-Item Measures of the Same Constructs. *Journal of Marketing Research* 44, 2 (May 2007), 175–184. https://doi.org/10.1509/jmkr.44.2.175 Publisher: SAGE Publications Inc.

[3] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: the growth of computer science. *ACM Inroads* 8, 2 (may 2017), 44–50. https://doi.org/10.1145/3084362

[4] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: the mixed news on diversity and the enrollment surge. *ACM Inroads* 8, 3 (jul 2017), 36–42. https://doi.org/10.1145/3103175

[5] Sarah E. Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping Distributed Hierarchical Web Data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST*

'18). Association for Computing Machinery, New York, NY, USA, 963–975. https://doi.org/10.1145/3242587.3242661

[6] Parmit K. Chilana, Celena Alcock, Shruti Dembla, Anson Ho, Ada Hurst, Brett Armstrong, and Philip J. Guo. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 251–259. https://doi.org/10.1109/VLHCC.2015.7357224

[7] Parmit K. Chilana, Rishabh Singh, and Philip J. Guo. 2016. Understanding Conversational Programmers: A Perspective from the Software Industry. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1462–1472. https://doi.org/10.1145/2858036.2858323

[8] Louis Cohen, Lawrence Manion, and Keith Morrison. 2018. *Research methods in education* (eighth edition ed.). Routledge, London New York.

[9] Kathryn Cunningham, Barbara J. Ericson, Rahul Agrawal Bejarano, and Mark Guzdial. 2021. Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code's Purpose. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3411764.3445571

[10] Kathryn Cunningham, Yike Qiao, Alex Feng, and Eleanor O'Rourke. 2022. Bringing "High-level" Down to Earth: Gaining Clarity in Conversational Programmer Learning Goals. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1 (SIGCSE 2022, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 551–557. https://doi.org/10.1145/3478431.3499370

[11] Zachary Dodds, Malia Morgan, Lindsay Popowski, Henry Coxe, Caroline Coxe, Kewei Zhou, Eliot Bush, and Ran Libeskind-Hadas. 2021. A Biology-based CS1: Results and Reflections, Ten Years In. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 796–801. https://doi.org/10.1145/3408877.3432469

[12] Brian Dorn. 2011. ScriptABLE: supporting informal learning with cases. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1145/2016911.2016927

[13] Brian Dorn and Mark Guzdial. 2010. Discovering computing: perspectives of web designers. In *Proceedings of the Sixth International Workshop on Computing Education Research* (Aarhus, Denmark) *(ICER '10)*. Association for Computing Machinery, New York, NY, USA, 23–30. https://doi.org/10.1145/1839594.1839600

[14] Brian Dorn and Mark Guzdial. 2010. Learning on the job: characterizing the programming knowledge and learning strategies of web designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 703–712. https://doi.org/10.1145/1753326.1753430

[15] Jacquelynne S. Eccles. 1983. Expectancies, values, and academic behaviors. In *Achievement and achievement motives*.

[16] Jacquelynne S. Eccles. 1987. Gender Roles and Women's Achievement-Related Decisions. *Psychology of Women Quarterly* 11, 2 (1987), 135–172. https://doi.org/10.1111/j.1471-6402.1987.tb00781.x _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1471-6402.1987.tb00781.x.

[17] Jacquelynne S. Eccles. 2005. Subjective Task Value and the Eccles et al. Model of Achievement-Related Choices. In *Handbook of competence and motivation*. Guilford Publications, New York, NY, US, 105–121.

[18] Jessica Kay Flake, Kenneth E. Barron, Christopher Hulleman, Betsy D. McCoach, and Megan E. Welsh. 2015. Measuring cost: The forgotten component of expectancy-value theory. *Contemporary Educational Psychology* 41 (2015), 232–244. https://doi.org/10.1016/j.cedpsych.2015.03.002

[19] Andrea Forte and Mark Guzdial. 2005. Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *IEEE Trans. on Educ.* 48, 2 (may 2005), 248–253. https://doi.org/10.1109/TE.2004.842924

[20] Dan Garcia, Brian Harvey, and Tiffany Barnes. 2015. The beauty and joy of computing. *ACM Inroads* 6, 4 (nov 2015), 71–79. https://doi.org/10.1145/2835184

[21] Mark Guzdial. 2003. A media computation course for non-majors. In *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education* (Thessaloniki, Greece) *(ITiCSE '03)*. Association for Computing Machinery, New York, NY, USA, 104–108. https://doi.org/10.1145/961511.961542

[22] Mark Guzdial. 2010. Does contextualized computing education help? *ACM Inroads* 1, 4 (dec 2010), 4–6. https://doi.org/10.1145/1869746.1869747

[23] Mark Guzdial. 2013. Exploring hypotheses about media computation. In *Proceedings of the ninth annual international ACM conference on International computing education research*. ACM, San Diego San California USA, 19–26. https://doi.org/10.1145/2493394.2493397

[24] Mark Guzdial. 2015. Learner-Centered Design of Computing Education: Research on Computing for Everyone. *Synthesis Lectures on Human-Centered Informatics* 8 (11 2015), 1–165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033

[25] Mark Guzdial and August Evrard. 2024. Identifying the Computing Education Needs of Liberal Arts and Sciences Students (Discussion Paper). In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '23)*. Association for Computing Machinery, New York, NY, USA, Article 19, 7 pages. https://doi.org/10.1145/3631802.3631805

[26] Mark Guzdial and Allison Elliott Tew. 2006. Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, United Kingdom) *(ICER '06)*. Association for Computing Machinery, New York, NY, USA, 51–58. https://doi.org/10.1145/1151588.1151597

[27] Meng Han, Zhigang Li, Jing He, and Xin Tian. 2019. What are the Non-majors Looking for in CS Classes?. In *2019 IEEE Frontiers in Education Conference (FIE)*. 1–5. https://doi.org/10.1109/FIE43999.2019.9028448 ISSN: 2377-634X.

[28] Amy J. Ko. [n. d.]. Amy J. Ko - CER FAQ. https://faculty.washington.edu/ajko/cer

[29] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The state of the art in end-user software engineering. *ACM Comput. Surv.* 43, 3, Article 21 (apr 2011), 44 pages. https://doi.org/10.1145/1922649.1922658

[30] Kathleen J. Lehman, Annie M. Wofford, Michelle Sendowski, Kaitlin N. S. Newhouse, and Linda J. Sax. 2020. Better Late Than Never: Exploring Students' Pathways to Computing in Later Stages of College. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1075–1081. https://doi.org/10.1145/3328778.3366814

[31] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[32] Ran Libeskind-Hadas and Eliot Bush. 2013. A first course in computing with applications to biology. *Briefings in bioinformatics* 14, 5 (2013), 610–617.

[33] Antonio M. Lopez, Marguerite S. Giguette, and Lisa J. Schulte. 2006. Large dataset offers view of math and computer self-efficacy among computer science undergraduates. In *Proceedings of the 44th Annual Southeast Regional Conference* (Melbourne, Florida) *(ACM-SE 44)*. Association for Computing Machinery, New York, NY, USA, 158–163. https://doi.org/10.1145/1185448.1185484

[34] Stephanie Lunn, Leila Zahedi, Monique Ross, and Matthew Ohland. 2021. Exploration of Intersectionality and Computer Science Demographics: Understanding the Historical Context of Shifts in Participation. *ACM Transactions on Computing Education* 21, 2 (June 2021), 1–30. https://doi.org/10.1145/3445985

[35] Bonnie A. Nardi. 1993. *A small matter of programming: perspectives on end user computing*. MIT press.

[36] Leo Porter and Beth Simon. 2013. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 165–170. https://doi.org/10.1145/2445196.2445248

[37] Catherine L. Saunders, Marc N. Elliott, Georgios Lyratzopoulos, and Gary A. Abel. 2016. Do Differential Response Rates to Patient Surveys Between Organizations Lead to Unfair Performance Comparisons?: Evidence From the English Cancer Patient Experience Survey. *Medical Care* 54, 1 (Jan. 2016), 45. https://doi.org/10.1097/MLR.0000000000000457

[38] Linda J. Sax, Jennifer M. Blaney, Christina Zavala, and Kaitlin N. S. Newhouse. 2020. Who Takes Intro Computing? Examining the Degree Plans of Introductory Computing Students in Light of Booming Enrollments. In *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*. IEEE, Portland, OR, USA, 1–7. https://doi.org/10.1109/RESPECT49803.2020.9272431

[39] Linda J. Sax, Kathleen J. Lehman, and Christina Zavala. 2017. Examining the Enrollment Growth: Non-CS Majors in CS1 Courses. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, Seattle Washington USA, 513–518. https://doi.org/10.1145/3017680.3017781

[40] Christopher Scaffidi, Shaw Mary, and Brad Myers. 2005. Estimating the Numbers of End Users and End User Programmers. *Proceedings - 2005 IEEE Symposium on Visual Languages and Human-Centric Computing* 2005. https://doi.org/10.1109/VLHCC.2005.34

[41] William G. Smith. 2008. *Does Gender Influence Online Survey Participation? A Record-Linkage Analysis of University Faculty Online Survey Response Behavior*. Technical Report. https://eric.ed.gov/?id=ED501717 Publication Title: Online Submission ERIC Number: ED501717.

[42] Elliot Soloway, Mark Guzdial, and Kenneth E. Hay. 1994. Learner-centered design: the challenge for HCI in the 21st century. *Interactions* 1, 2 (apr 1994), 36–48. https://doi.org/10.1145/174809.174813

[43] Koen J.F. Verhoeven, Katy L. Simonsen, and Lauren M. McIntyre. 2005. Implementing false discovery rate control: increasing your power. *Oikos* 108, 3 (2005), 643–647.

[44] April Y. Wang and Parmit K. Chilana. 2019. Designing Curated Conversation-Driven Explanations for Communicating Complex Technical Concepts. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 211–215. https://doi.org/10.1109/VLHCC.2019.8818822 ISSN: 1943-6106.

[45] April Y. Wang, Ryan Mitts, Philip J. Guo, and Parmit K. Chilana. 2018. Mismatch of Expectations: How Modern Learning Resources Fail Conversational Programmers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174085

[46] Dana Wanzer, Tom McKlin, Doug Edwards, Jason Freeman, and Brian Magerko. 2019. Assessing the Attitudes Towards Computing Scale: A Survey Validation Study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, Minneapolis MN USA, 859–865. https://doi.org/10.1145/3287324.3287369

[47] Allan Wigfield. 1994. Expectancy-value theory of achievement motivation: A developmental perspective. *Educational Psychology Review* 6, 1 (March 1994), 49–78. https://doi.org/10.1007/BF02209024

[48] Mingjing Zhu and Detlef Urhahne. 2014. Assessing teachers' judgements of students' academic motivation and emotions across two rating methods. *Educational Research and Evaluation* 20, 5 (July 2014), 411–427. https://doi.org/10.1080/13803611.2014.964261 Publisher: Routledge.