# Perceptions of Non-CS Majors in Intro Programming: The Rise of the Conversational Programmer

Parmit K. Chilana[1], Celena Alcock[1], Shruti Dembla[1], Anson Ho[1], Ada Hurst[1], Brett Armstrong[1], and Philip J. Guo[2]
[1]University of Waterloo
Waterloo, ON, Canada
[2]University of Rochester
Rochester, NY, USA

*Abstract*— Despite the enthusiasm and initiatives for making programming accessible to students outside Computer Science (CS), unfortunately, there are still many unanswered questions about how we should be teaching programming to engineers, scientists, artists or other non-CS majors. We present an in-depth case study of first-year management engineering students enrolled in a required introductory programming course at a large North American university. Based on an inductive analysis of one-on-one interviews, surveys, and weekly observations, we provide insights into students' motivations, career goals, perceptions of programming, and reactions to the Java and Processing languages. One of our key findings is that between the traditional classification of non-programmers vs. programmers, there exists a category of *conversational programmers* who do not necessarily want to be professional programmers or even end-user programmers, but want to learn programming so that they can speak in the "programmer's language" and improve their perceived job marketability in the software industry.

**Keywords— Computer science education; computational thinking; programming for non-CS majors**

## I. INTRODUCTION

The growing demand for a technology-savvy workforce in the 21st century has stirred a number of debates[1] around how to best equip college graduates with computational thinking and computer programming skills. Some universities have introduced special mandates for teaching programming to non-Computer Science (CS) majors in specific disciplines, such as Science, Engineering, and Arts [9]. Other institutions (e.g., Georgia Tech) have gone as far as requiring *all* non-CS majors to enroll in at least one programming course to satisfy graduation requirements [11].

Despite the enthusiasm and growing initiatives for making programming accessible to everyone, unfortunately, there are many unanswered questions about what actually works or does not work in a classroom of engineers, scientists, artists or other non-CS majors [14]. One root problem is our lack of understanding of the non-CS university student population, which is far larger and less researched than the population of CS majors. Guzdial [14] argues that most of our intuitions and assumptions about teaching programming are based on experiences of first-year CS courses: so, what are the actual needs, perceptions, motivations, and learning strategies of non-CS students who are taking programming courses?

[1] http://www.nytimes.com/2012/04/01/business/computer-science-for-non-majors-takes-many-forms.html

Another open question is about the perceived benefits of programming skills among non-CS students. One prevalent assumption is that non-CS majors, particularly in STEM fields, will likely become *end-user programmers* and write programs to support their domain-specific work [3,17,24]. But, is that necessarily the case for *all* non-CS students? Or is there some other value proposition for learning programming if the desired goal is neither to become a professional programmer nor an end-user programmer?

In this paper, we investigate these questions from the perspective of non-CS majors enrolled in management engineering, an interdisciplinary engineering undergraduate program. This population is intriguing because unlike students in biology, business, or fine arts, management engineers receive training that crosses boundaries of multiple disciplines, including industrial engineering, management information systems, operations research, and behavioral sciences. Since these types of interdisciplinary undergraduate programs are becoming increasingly prevalent (particularly with the emerging *Informatics* and *Data Science* movements [23]), it is important to understand how the enrolled students' goals and perceptions are shaped by exposure to programming.

The research site for our study was a required first-year programming course for management engineering students taught in its home department. The 13-week course introduced basic programming concepts through the *Processing* language for the first two weeks and then focused on the fundamentals of *Java*. We used a mixed-method approach for our data collection, carrying out 25 one-on-one interviews, 2 surveys at the beginning and near the end of the course, respectively, and 10 sessions of 2-hour classroom observations. We focused our investigation on understanding management engineering students' motivations, career goals, perceptions of programmers and programming, and reactions to code.

One key finding from our study was that even though only 7% of students listed "programmer" as an ideal career choice, over 73% of students indicated interest in continuing to learn programming beyond this first course. Not surprisingly, half of these management engineering students wanted to develop skills for being able to work on end-user programming tasks (such as data analysis and project management). However, further analysis of our data showed two other trends as to why the other students valued programming literacy: 1) to

understand the work of professional programmers and establish common ground in communication and, 2) to increase the perceived marketability of their skills for future internships and jobs. We characterize this subgroup as *conversational programmers*—students who want to be literate in programming not for the sake of becoming professional or end-user programmers, but for the pragmatic reason of being able to converse in the "programmer's language" and improving their perceived marketability in the software industry. Our initial results suggest that these conversational programmers perceive industry-standard languages (e.g., Java) to be more beneficial than visual or domain-specific teaching languages (e.g., Processing) that simplify programming syntax.

Our main contribution in this paper is in providing empirical evidence that shows that among the traditional classification of non-programmers vs. end-user programmers vs. professional programmers, there exists a category of conversational programmers who fall between the spectrum of non-programmers and end-user programmers and who want to learn to "speak the programmer's language". Given the diversity of non-CS students coming into introductory programming classrooms, there is a greater need to better understand these students' perceptions and enthusiasm for programming. In our discussion, we tackle the question of how we should be thinking about training conversational programmers and non-CS majors in general and how we can strike a balance between teaching programming for intellectual enrichment vs. job marketability.

## II. RELATED WORK

Despite decades of research in computing education, the existing literature mostly focuses on the experiences of teaching programming to first-year CS students—there are few direct investigations of the experiences of non-CS majors. Still, many recommendations have been made for simplifying introductory programming courses that are relevant for non-CS majors.

**Lightweight introductory CS courses:** Many languages have been used throughout the years for teaching introductory CS courses (Java, C, and C++ top the list of the most widely used programming languages in the last two decades [21,25]). There have been many debates and concerns around how to balance the need to teach such mainstream languages without adding extra complexity to introducing CS concepts, especially for novices [5,15,20]. One approach advocated for teaching non-CS students is to take a subset of the content from a course for CS majors [9,10,18,26] and create a "lightweight" introduction. This approach simplifies the intro CS content by focusing on programming fundamentals, while still introducing a widely-used language. Although there is some evidence that tailoring traditional intro CS courses to non-CS majors can be successful, we do not know if this applies to all categories of non-CS majors or if some non-CS majors are more likely to benefit. The programming course that is the subject of our study used the approach of creating a "lightweight" introduction (using Processing), while still using Java as the main programming language and curriculum adapted from an introductory CS course.

**Visual languages that simplify programming:** Instead of teaching mainstream programming languages, some researchers argue for the use of *visual programming* to simplify the teaching of complex concepts. Visual programming languages allow users to visually demonstrate or sketch their program flow, rather than using commands, pointers, and abstract symbols [1, 2]. Visual programming environments based on storytelling metaphors, such as *Scratch* [22] or *Alice* [16], have been shown to be particularly successful at introducing basic programming concepts by letting students create multimedia animations. Similar recommendations have been embodied in efforts such as *RAPTOR* [4] that uses flowcharts to demonstrate complex concepts and *App Inventor* [28] that focuses on visual mobile application development.

A recent trend has been to use the Processing[2] language in intro programming courses. Processing is an open-source language derived from Java that is designed to allow programmers to easily add graphics, animation, media, and user interaction to their programs. This language has been particularly appealing for teaching first-year students [19] because of the simplicity in setting up the development environment and creating interactive programs. Processing offers many built-in methods that allow students to tackle a variety of problems without getting too caught up in programming logic and syntax early on. In our study, we shed light on how the participants reacted to Processing, the transition to Java later in the course, and some of the tensions around ease-of-learning vs. future marketability.

**Teaching programming in context:** Another teaching approach for non-CS majors has been to create completely new courses that focus on programming in the context of other computational tasks For example, in the media computation course at Georgia Tech [3,11,13], Arts students get an introduction to programming by working with popular media—manipulating filters on images, editing sounds in audio files, writing scripts to extract content from the Web, creating animations, and so forth. In the *software carpentry* initiative [27], scientists are taught basic computing skills and use of command-line scripting tools to make them more efficient in scientific computation. There have also been efforts to teach domain-specific programming to experts such as professional designers who often learn programming skills "on the job" in the context of their daily design work [7]. The underlying assumption of teaching programming "in context" is that students work on end-user programming tasks in their respective domains. However, as we found in our study, even though most of the students wanted to take other programming courses, a portion of the management engineering students

[2] http://www.processing.org

were interested only in programming literacy and not in becoming end-user programmers.

In summary, prior work has made a number of recommendations for simplifying intro CS curricula and programming languages to accommodate the needs of novice programmers and non-CS majors, but the recommendations have not necessarily taken into account the diversity within the non-CS student population. Furthermore, with the rise of interdisciplinary programs, we can no longer assume that what works for a particular group of scientists, artists, or engineers will necessarily be applicable to students who are enrolled in interdisciplinary programs. Our case study adds detailed insights into the perceptions, reactions, and goals of interdisciplinary management engineers and how they differ from previous characterizations of non-CS majors in intro programming.

## III. RESEARCH SITE AND METHOD

### *A. Research Site*

We conducted this study in an *Introduction to Computer Programming* course offered as part of the management engineering undergraduate program at a large North American university. Management engineering is an emerging discipline that concerns the engineering (i.e., designing, planning or operating) of management systems. The program can be viewed as a modern form of traditional industrial engineering, with the new take reflecting (in part) the all-encompassing use of information systems in contemporary organizations.

The management engineering curriculum incorporates topics pertaining to operations management, logistics and supply chain management, inventory control, economics, accounting, organizational studies, and design of information systems [8]. All first-year students in the management engineering program are required to take the introductory programming course taught in the home department. All students in the program are also required to participate in the cooperative education internship program (co-op) for at least five terms during their study. Students can earn this experience in a variety of industries, with a majority of management engineering students typically working in the manufacturing, high-tech, retail, and financial industries.

The course that was the subject of this study introduced programming fundamentals in the Processing language for the first 2 weeks and transitioned to Java for the remaining 11 weeks. The course topics included basic components of algorithms, primitive operations, variables, sequencing operations, conditionals and branching, subroutines, problem decomposition, abstraction, file-based input and output, use of a modern development environment, pointers/references, and basic data structures, such as arrays.

### *B. Study Methods*

We collected data for this study using three different methods:

**Surveys:** Two of the authors distributed two survey questionnaires to students enrolled in the course: the first survey was distributed at the beginning of the course (phase 1) and the second near the end of the course (phase 2). The survey was voluntary and did not affect student grades nor did the instructor have any knowledge of who participated. The surveys tried to capture the students' perceptions of the course and programming, and their overall career goals. At the end of the surveys, students could opt-in to participate in a brief interview in two different intervals.

The survey in phase 1 consisted of questions about prior programming experience and reactions to the introduction to Processing. The survey in Phase 2 focused on Java and the transition from Processing to Java. It included questions about how confident students were using the Java programming language and their reactions to syntax. The surveys also had questions about how useful they thought it was to learn more programming languages, if they would consider taking more programming courses, what they would do with programming, their ideal future jobs and career goals related to their major.

We received 51 responses to the survey in Phase 1 (response rate of 69%) and 56 responses to the survey in Phase 2 (response rate of 75%). We attribute the high response rate to the surveys being short and distributed on paper during two of the tutorial sessions where students often had downtime between in-class exercises.

**One-on-one interviews:** Based on the survey responses, two of the authors recruited 12 interviewees for Phase 1 of the study, and 13 interviewees for Phase 2 (both phases had separate sets of interviewees). The interviews lasted around 30 minutes on average and were semi-structured in format.

Both rounds of interviews focused on participants' prior programming experiences, their perceptions about programming before and after taking the course, their reactions to learning Processing vs. Java, what the students wanted to do for their co-op internships in industry, what they want to do after graduation, and their perceptions around the value of learning more programming.

**Weekly observations:** Four of the authors carried out weekly observations during a 2-hour tutorial session for the course. These observations were unstructured: the authors took detailed de-identified field notes in their notebook about students' participation in the tutorial activities, what questions they were asking, where they were having confusions or difficulties, and when they were asking for help.

### *C. Analysis and Presentation of Results*

We audiotaped and transcribed all of the interviews and observation notes. All transcripts were organized, coded, and analyzed using the *NVivo* data analysis software. We used a bottom-up inductive analysis approach to explore different facets of the students' narratives and identified recurring themes.

Since our interviews, surveys, and observations produced a large amount of data, we present our results in terms of the major themes that emerged.

## IV. WHO ARE MANAGEMENT ENGINEERS?

We first present the overall demographics of our student population, since it differs not only from a typical CS major population that has been well studied, but also from traditional non-CS students due to the interdisciplinary training that management engineers receive. Furthermore, since these students were expected to start their first co-op work term at the end of their first year, we found that most of them were already aware of how they would need to market their skills for competitive co-op positions (which is typically not the case for undergraduates in other programs without co-op).

In our interviews, we first probed into why students enrolled into the management engineering program. Many of them said they found the interdisciplinary nature of the program to be appealing. For example, one participant (P12) explained:

> *It [the program] lets me experiment with a bunch of different fields…I never knew what kind of specific engineering field I wanted to narrow down to…while doing management engineering, it's like I'm learning a bit of everything from every other field and at the end I have to work with multi disciplinary teams…(P12)*

In our survey, respondents listed a number of different career paths as their ideal choice. We classified the free-form responses into 9 higher-level categories (top 6 categories that received at least 5% of responses are shown in Figure 1). Most management engineering students were enthusiastic about taking on jobs such as a project manager, becoming a business executive, such as a CEO, or owning a business. Only 7% listed programmer as an ideal career choice. (Note that 29% of respondents did not answer this survey question or wrote that they were not sure). These expectations bear a very limited resemblance to actual post-graduation employment. In particular, over 70% of this program's graduates work as management consultants or business analysts[3]. These roles can be in a variety of contexts including high-tech and supply chain. We do not know how much end-user programming is included in these positions, but we do know that some of these jobs have involved end-user programming. About 10% of the graduates work in project/program/product management roles. Only a small minority (~7%) become full-time programmers, and an even smaller minority (3%) become entrepreneurs. The remaining graduates fulfill traditional industrial engineering positions or go on to graduate studies.

According to our survey data, 72% of the students had no formal exposure to programming before taking this course and 91% were taking a formal course in Java for the first time. (12% of students mentioned getting exposure to languages such as C#, Visual Basic, and Python in high school). Our interviews revealed that the remaining participants got

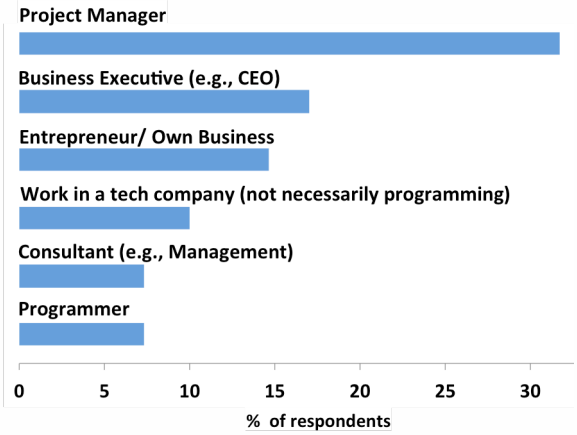


**Figure 1: Classification of survey responses into the top 6 categories of "ideal jobs" listed by our survey respondents**

exposure to programming by studying online tutorials over summer or winter breaks.

Throughout the study, we found that the students' disciplinary training and the co-op program both heavily influenced their perceptions, self-efficacy, and value judgments related to learning Processing, Java, and programming in general.

## V. PERCEPTIONS OF PROGRAMMERS AND PROGRAMMING

One goal of our study was to understand the perceptions that students had about programmers and programming prior to taking this first-year class. We saw two themes in the responses: avoiding programming out of fear of the perceived learning challenges and simply not being aware of what programing actually entailed.

For example, one of our interviewees who chose not to take a programming course in high school explained his rationale:

> *I wanted to take [programming] in grade 11 [and] grade 12 but I feared that I wasn't capable to think like a computer and I heard that the assignments that they gave were challenging and people kind of just struggled with it…I didn't want to struggle…(P11)*

Another theme we observed in the responses was misconceptions and generalizations about what programmers actually do at work:

> *When you think of programming you think of someone like sitting in a dark room typing on a keyboard all the time…(P06)*

Many students explained that social media and how programmers are portrayed in popular culture contributed to their perceptions of programming:

> *I just thought programming was mainly used in the hacking field, for example…because you know watching movies and stuff, right, you see oh they're shutting down street lights and stuff…(P13)*

However, within the first few weeks of the course, students started discovering that programming involved a lot of

[3] https://uwaterloo.ca/management-sciences/future-undergraduate-students/why-management-engineering

problem solving and there was more to it than just *"typing in a dark room:"*

> *You don't have to be amazingly smart to do it but like you do have to have the skill of breaking down problems which is something I've learned now but before I thought it was only a subject that a certain few could be good at…(P25)*

Students felt that understanding what programming is and how it actually works helped them debunk some of their earlier misconceptions about the utility of programming:

> *…basically everything runs on programming, you've [got] your account, files, everything stored, even learning…basically the world is connected on computers now, and programming runs all our social lives, academic lives and professional lives. It runs everything. What would we do without programming? (P12)*

Even though our participants were accustomed to having a heavy engineering workload and challenging math and science courses, they found programming to be a lot different from their other courses:

> *It's a different type of problem solving I guess. Like in engineering a lot of times you might be battling with restraints and not having enough resources but this is kind of like analytical thinking about how you can approach stuff differently…(P04)*

Some students mentioned that after taking the course they realized that they would benefit from programming in their everyday life. They realized that programming was a skill they could carry with them everywhere:

> *Just the problem solving techniques that you learn I think are stuff that you apply even in like day-to-day problem solving even if you don't realize it…like you're not writing code when you have an issue on a day-to-day basis but the problem solving techniques and the steps that you go through still apply in real life… (P19)*

### *A. Why Learn Programming If You're Not Planning on Becoming a Programmer?*

One salient finding from our surveys was that despite the perceptions that students initially had and some of the frustrations that they described in learning programming for the first time, over 73% of the respondents *wanted* to take another programming course after this intro course. When we asked respondents to list the kinds of things they can imagine using programming for in the future (either for work or hobby), 48% of respondents listed future work-related tasks involving end-user programming, such as data analysis, process automation, solving calculations, and making mobile apps, among others. (Interestingly, none of the responses indicated anything related to pursuing a hobby.)

In our interviews, we probed into understanding why students valued programming and wanted to take more programming courses (if it was not for the reason of pursuing end-user programming). The most common reason was that early on in the program, students had formed a perception that programming was an "*important skill to have on your resume"*, and "*a necessary evil" (*based on what they had heard about the co-op program from other students).

> *…it became apparent that you had to know some type of programming language...it was like a staple of your intelligence knowing that if you knew how to program …like you need something to attract an employer and it seemed like everybody was mentioning programming so that was something that you kind of had to look into even if you didn't really want to…I'm enjoying it [programming] even though like again it's tedious but the reason I first looked into it was because it seemed like you needed it to, to even apply to any [co-op] job basically…(P25)*

Increasing the potential for securing job opportunities was one of the most common responses among the participants:

> *I guess it [programming] just like broadens [the] range of skills that you have…so, say I know a lot about management science but I also know programming which can help me in say like, if I'm looking for a job I'll have like experience in Java as well as all my other experiences. It's just like to add on to like the list of skills…(P16)*

This sentiment was echoed by a number of participants:

> *I'd just learn some other programming language because it looks good on the resume like I don't really enjoy programming...I think more of a qualification part would be for me to learn programming languages…(P22)*

We also found that those students who did not want to take another formal course were still willing to learn on their own, at their own pace:

> *I would like to learn more programming…it's useful looking for co-op jobs for sure. But it's just the timing and it takes up a lot of time. I think it's, there's so many resources on like the Internet to learn but I guess there's just no time for it. If I was on a co-op term I think I would have more time to actually learn so I think I might pick up some things like that...(P05)*

On the flip side, we also probed into why over a quarter of the students did *not* want to take another programming course. In this case, the common response was, "*it's not for me*:"

> *It [programming] requires a lot of patience and it requires a person to dig in to figure out what's going on. I'm not that type of person. I do math easily, but digging in and trying to invent something, that's what I want to do but I'm not that person by myself...(P17)*

In summary, many students learned that programming was more about problem solving (and not just cryptic typing) and could see the application of programming skills beyond the classroom. Interestingly, even though more than half of the students did not want to be professional programmers or even

end-user programmers, they valued programming literacy for being able to market themselves for future jobs.

## VI. PERCEPTIONS OF SELF-EFFICACY AND COMPTENCE WHEN PROGRAMMING

As mentioned earlier, a number of the participants did not know what programming was or were afraid to try it in high school. However, once they started taking a programming course, their self-confidence started to improve. Our surveys showed that after being introduced to Processing, over 70% of students felt confident about writing short programs, and after learning Java, over 50% of students felt confident about writing short programs in Java (Figure 2). Being able to work on actual programming problems and seeing results demystified the assumptions that students had made about programming and their self-efficacy:

> *I always wondered how computers worked and I always thought it was so beyond my level of knowledge that I would never be able to learn that…so, the fact that I actually am learning that, it's really interesting…(P07)*

Many of the students wanted to keep learning programming to be more self-confident when working in the software industry in business or management positions:

> *If we do anything [in] management we don't want to be standing there talking to someone not having a clue what they're talking about. We should at least…I think, I don't know, I couldn't do your [software engineering] job but when you're talking to me I know what you're saying and I don't feel totally like an idiot…(P14)*

Our participants not only appreciated the value of being able to communicate with programmers, but also in being able to better understand and appreciate programmers' efforts:

> *I have much more respect for people who do programming for a living, I certainly couldn't. Maybe if it was my job I would be able to because I could put in the hours for it, but right now it's really humbling when you think of how some people can create incredible pieces of work and I'm struggling to do assignment questions or what not. I really just gained a huge amount of respect for people who can do it and can do it well…(P24)*

Despite the enthusiasm for having programming as a marketable skill on resumes and confidence in writing short programs in Java and Processing, we discovered in our interviews that most students were not confident about actually pursuing programming jobs:

> *I don't think I'd be able to keep up with it [programming] very well, just I'm not the best programmer… they'd probably want me to do things that programming position wants me to do but it'd be a struggle, it'd be a huge struggle. I'd have to put in lots of extra hours to make sure I do the job...(P24)*

One of the main reasons for this lack of self-confidence was that the students worried about where they stood amongst their

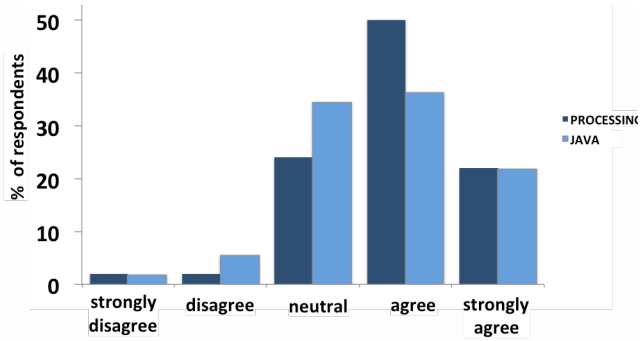


**Figure 2: Comparison of survey responses to the question, "*I am confident about writing short programs on my own*" Processing (left) vs. Java (right)**

peers in CS and software engineering programs, who also compete for the same programming jobs:

> *I think we're definitely capable of doing programming jobs but software engineering [major] is quite a bit ahead of us so I'd rather have someone else like that do it [programming]. I don't know if I'd feel entirely comfortable doing it…(P4)*

Even though the students were taught an introductory curriculum based on mainstream intro courses in Java, the students consistently demonstrated low self-efficacy because they assumed that other students would know more or have more experience:

> *Unless I devote an exceptional amount of my time, boring, ridiculous amount of my time above and beyond my own program I will never have the same level of experience or competence with the computer engineering and software students…(P15)*

In summary, even though students were enthusiastic about learning programming beyond this course and having programming experience on their resume, they were not as confident in *actually* taking up a programming job.

## VII. REACTIONS TO PROGRAMMING IN PROCESSING AND JAVA

In trying to better understand students' assessments of their self-efficacy and competence, we probed into students' reactions to programming for the first time and how they viewed the transition from Processing to Java in the course.

In relation to both programming languages, students described a number of issues related to syntax, logic, debugging and "*thinking like a computer*", consistent with previous studies [20] on novice learners. During the weekly observations, we observed that the majority of students would start typing code before thinking of a solution to the problem. Particularly when students started working on Java after learning Processing, many of them initially struggled with problem decomposition, and often relied on trial and error. Our interviews also confirmed these findings:

> *Your critical thinking has to be really strong so I feel like that's the hard part about Java…you have to know how to start the problem, you have to know the scenarios and the*

*cases you're going to pick to actually make the program work instead of just typing it out…(P13)*

Students who were coding in Java for the first time were most frustrated by the Eclipse IDE and not being able to understand why their program would sometimes compile and fail at other times:

*Computers are a lot pickier with their language than I thought…they don't think like a person reading it would think, they think very literal. It's kind of like talking to someone who's mocking you by responding completely literally to what you mean…(P14)*

Despite some of the frustrations, students also shared stories of resilience and improvement throughout the term:

*You don't have to be a genius but you do have to persevere a lot, especially when you're trying to find that one semi-colon that's making your program not compile... yeah perseverance and not breaking your computer…(P25)*

When we asked students to compare their reactions to learning Processing vs. Java in the course, our surveys indicated that 80% of students felt that it was useful to learn Processing before Java. The interview responses were consistent with this result:

*I thought it was good that we started out with Processing because for someone who has had no programming experience it was kind of like we're easing you into it, we're not like throwing you in the deep end right from the get go. (P25)*

From a learning perspective, participants consistently appreciated the graphical aspect of Processing and its simple IDE compared to the initial programs they wrote in Java:

*It's nice to be able to see an output [in Processing], to program something and immediately see exactly what it did and…but, in Java [so far] we have only done things that output a word into the bottom of the console…whereas with Processing it would be a separate thing, you could output something to the console or you could create your own thing…(P01)*

Interestingly, even though students found Java more challenging to learn than Processing, their common perception was that Java was more useful:

*I think Java's more practical. Ah Processing might be more fun like you can make shapes and draw pictures and stuff like that but I don't think it's really useful…(P4)*

*I think Processing is used for more just graphic designs like animations and that kind of stuff…I think Java will be more complex and so we'll learn more, I think Java's a lot more useful…(P5)*

There was also a perception (or perhaps misconception) that Processing was not useful for anything in the real world:

*I think, to me it feels like Processing is a very old kind of programming language…if you think that in 70s when there were still those big computers and all they could do was like output words, I think that's my connection between Processing and those computers, doesn't really do much…(P11)*

Although students did not provide any clear reasons why they thought Processing was not useful, most of their perceptions seemed to have been influenced by not only the nature of the course assignments, but also the number of job postings related to Java:

*I feel like Java would be more applicable but I feel Processing for me was more [enjoyable], because of the more visual artistic part…that's something I'd enjoy more working with…I'd prefer Java because like we haven't done a lot of Processing and really…like how much farther can you take it? And companies recognize Java a lot more than Processing…(P21)*

In summary, many of the struggles and frustrations that students described with initial exposure to Processing vs. Java were not surprising, given the similarity to findings of previous studies on novice programmers [20]. However, the surprising finding was the kinds of perceptions that these first-year students formed about the utility and usefulness of languages based on their *marketability*, rather than learnability.

## VIII. DISCUSSION

We have presented a detailed case study of first-year programming in a management engineering program where students receive interdisciplinary training in engineering, science, and management, and are required to complete mandatory co-op internships. Our study adds to existing studies of non-CS majors, highlighting students' unique motivations, career goals, perceptions of programmers and programming, and reactions to different languages. Although our study is limited to one population, we discuss the importance of some of the initial evidence and potential implications for pedagogy and computing education research.

### *A. Rise of the "Conversational Programmer"*

Our results indicate that while many of the management engineers were interested in being end-user programmers (similar to [3,17,24]), some of the students wanted to develop only "conversational" skills in programming so that they can communicate with programmers in the future and improve their perceived marketability in the software industry (e.g., similar to being conversationally proficient in a foreign language). This finding suggests that among the traditional classification of non-programmers vs. end-user programmers versus professional programmers, there exists a category of students who want to be literate in programming to converse in the "programmer's language." We characterize a person who is more programming literate than a non-programmer, but not necessarily an end-user programmer, as a *conversational programmer* (Figure 3).

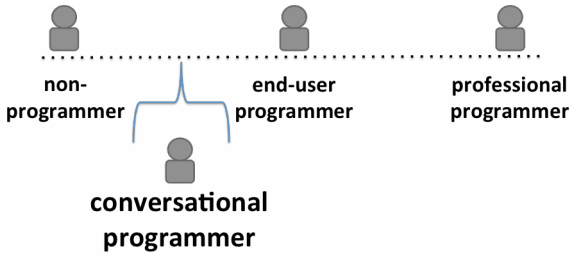


**Figure 3: Our results point to the presence of *conversational programmers*, who are interested in becoming programming literate, but not necessarily end-user programmers**

Given the rise of interdisciplinary undergraduate programs similar to management engineering (e.g., that bridge business, data manipulation, computational science, and engineering skills), it would not be surprising to see conversational programmers emerge in other non-CS disciplines as well.

### *B. Pedagogical Dilemmas for Conversational Programmers*

So, should we be teaching conversational programmers differently? We found that conversational programmers were well aware of the utility of mainstream programming languages (such as Java) and perceived them to be more useful than teaching-oriented languages (such as Processing), partly due to the mandatory co-op program. This creates an interesting tension in this type of a non-CS intro programming classroom: how much emphasis should we give to visual languages to achieve better learning outcomes vs. teaching mainstream languages for the sake of marketability? Similarly, should we keep exploring special IDEs and tools to support the learning needs of novices and non-CS majors (e.g., with [4]), or should we expose them to realistic professional IDEs for better exposure to industry-level software development?

Also, when we have a classroom where non-CS majors place different value judgments associated with learning programming, how applicable are domain-specific or specialized courses [6,12]?

For over a decade, we have known that the population of end-user programmers far exceeds professional programmers [24]. Consistent with prior findings, we also found that half of the students in our study wanted to work on end-user programming tasks, such as data analysis and project management (using spreadsheets). However, we cannot assume that *all* non-CS majors learn programming for the sake of becoming end-user programmers. Is it still worth it to instill strong end-user programming skills, even if conversational programmers may never actually write any code? We believe there is a rich space to further explore such questions about the pedagogy of conversational programmers.

### *C. Limitations and Future Work*

Despite some of the new research questions that our study opens up, it has several limitations.

First, our findings represent the perspective of only one particular undergraduate major—it may be that these findings are limited in other contexts. It will be valuable to have other such case studies in different disciplines to overall better understand the perspectives of non-CS majors in intro programming courses.

Also, since the students in our study were in their first year of the program, it is likely that their perspectives will change over time. We have not provided an explicit comparison to these perspectives in this paper because we wanted to first understand the initial perspectives of students that are formed upon their first exposure to code (and even before starting the intro programming course). Doing a comparison between first-year students and those who are, for example, closer to graduation would be an interesting direction for future work.

Finally, our findings may be limited to our institution and its unique interdisciplinary nature and the co-op program. Does the conversational programmer concept manifest in other contexts outside the university and other disciplines? What is the industry perspective on conversational programmers? We believe that these questions need to be addressed in future work and that there is merit in further exploring and developing the concept of conversational programmers.

## IX. CONCLUSION

For over two decades, we have seen several initiatives that introduce new ways of teaching programming and computational thinking skills to students in disciplines outside CS, but our knowledge of the non-CS population and their perceptions of programming is limited. In this paper, we have presented an in-depth study of one non-CS population of management engineering students enrolled in a required introductory programming course. Our study is among the first to shed light on the perceptions, motivations, and reactions to first-year programming from the perspective of these interdisciplinary non-CS students. Our findings suggest that successful domain-specific efforts that abstract out complex concepts and focus on what end-user programmers need (e.g., those in science vs. art vs. interdisciplinary programs, etc.) should be developed further since many non-CS students are likely to benefit from them. However, at the same time, we have to be open to other types of programming needs and value judgments among non-CS majors. The sub-population of conversational programmers that we discovered among management engineering students may just be the beginning—perhaps, there are other categories as well along the spectrum of non-programmer to professional programmer that need to be explored further.

## ACKNOWLEDGMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERNCES

[1] Burnett, M.M. 1999. Visual programming. In *Wiley Encyclopedia of Electrical and Electronics Engineering*. (John G. Webster, ed.) New York, NY: John Wiley & Sons.

[2] Burnett, M.M., Baker, M.J., Bohus, C., Carlson, P., Yang, S. and Van Zee, P. 1995. Scaling up visual programming languages. *Computer*, 28, 3, 45–54.

[3] Burnett, M.M. and Myers, B.A. 2014. Future of end-user software engineering: beyond the silos. *Proceedings of the on Future of Software Engineering*, 201–211.

[4] Carlisle, M.C., Wilson, T.A., Humphries, J.W. and Hadfield, S.M. 2004. Raptor: introducing programming to non-majors with flowcharts. *Journal of Computing Sciences in Colleges*, 19, 4, 52–60.

[5] Close, R., Kopec, D. and Aman, J. 2000. CS1: perspectives on programming languages and the breadth-first approach. *Journal of Computing Sciences in Colleges*, 228–234.

[6] Cooper, S. and Cunningham, S. 2010. Teaching computer science in context. *ACM Inroads*, 1, 1, 5–8.

[7] Dorn, B. and Guzdial, M. 2010. Learning on the job: characterizing the programming knowledge and learning strategies of web designers. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 703–712.

[8] Duimering, R., Elhedhli, S., Jewkes, B. and Smucker, M. 2013. *Management Engineering: The Engineering of Management Systems*. Available at: https://uwaterloo.ca/management-sciences/sites/ca.management-sciences/files/uploads/files/Management_Engineering_discussion_paper.pdf

[9] Forte, A. and Guzdial, M. 2005. Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *IEEE Education,* 48, 2, 248–253.

[10] Goldman, K.J. 2004. A concepts-first introduction to computer science. *ACM SIGCSE Bulletin,* 432–436.

[11] Guzdial, M. 2003. A media computation course for non-majors. *ACM SIGCSE Bulletin*, 104–108.

[12] Guzdial, M. 2010. Does contextualized computing education help? *ACM Inroads*. 1, 4, 4–6.

[13] Guzdial, M. and Forte, A. 2005. Design process for a non-majors computing course. *ACM SIGCSE Bulletin*, 361–365.

[14] Guzdial, M. 2015. Computing Education Must Go Beyond Intuition: The Need for Evidence-Based Practice. *BLOG@CACM (February 22, 2015)*.

[15] Hadjerrouit, S. 1998. Java As First Programming Language: A Critical Evaluation. *ACM SIGCSE Bulletin,* 43–47.

[16] Kelleher, C., Pausch, R. and Kiesler, S. 2007. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1455–1464.

[17] Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H. and Myers, B. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*. 43, 3, 21.

[18] Marks, J., Freeman, W. and Leitner, H. 2001. Teaching applied computing without programming: a case-based introductory course for general education. *ACM SIGCSE Bulletin*, 80–84.

[19] Meysenburg, M.M. *Introduction to Programming Using Processing*. 2014. Crete, NE: Mark M. Meysenburg.

[20] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. and Paterson, J. 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, 204–223.

[21] De Raadt, M., Watson, R. and Toleman, M. 2004. Introductory programming: what's happening today and will there be any students to teach tomorrow? *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*, 277–282.

[22] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J. and Silverman, B. 2009. Scratch: programming for all. *Communications of the ACM*. 52, 11, 60–67.

[23] Ryan, S. 2012. *Colleges with Data Science Degrees*. Available at: http://101.datascience.community/2012/04/09/colleges -with-data-science-degrees

[24] Scaffidi, C., Shaw, M. and Myers, B. 2005. Estimating the numbers of end users and end user programmers. *IEEE Visual Languages and Human-Centric Computing,* 207–214.

[25] Stephenson, C. and West, T. 1998. Language choice and key concepts in introductory computer science courses. *Journal of Research on Computing in Education*, 31, 1, 89–95.

[26] Urban-Lurain, M. and Weinshank, D.J. 2000. Is there a role for programming in non-major computer science courses? *Frontiers in Education Conference, 1,* T2B/7–T2B11.

[27] Wilson, G. 2006. Software carpentry. *Computing in Science & Engineering*, 8, 66-69.

[28] Wolber, D. 2011. App inventor and real-world motivation. *Proceedings of ACM technical symposium on Computer science education*, 601–606.