Novice Reflections During the Transition to a New Programming Language

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Nigel Bosch
University of Illinois
Urbana–Champaign, Illinois, USA
pnb@illinois.edu

James Prather
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Brent Reeves
Abilene Christian University
Abilene, Texas, USA
brent.reeves@acu.edu

Jacqueline Whalley
Auckland University of Technology
Auckland, New Zealand
jacqueline.whalley@aut.ac.nz

ABSTRACT

As computing students progress through their studies they become proficient with multiple programming languages. Prior work investigating language transitions for novices has tended to analyze program artifacts rather than explore the benefits and difficulties as perceived by students in their own words, and has often overlooked problems that may arise in switching paradigms or where familiar syntax has a different meaning in the new language. In this paper, we ask students to reflect on the transition from an interpreted language and environment (MATLAB) to a compiled language (C), prompting comments on the aspects of learning the new language that they found both easier and harder. Analysis of over 70,000 words written by 771 students revealed that the highest-performing students expressed more negative sentiments towards the language transition – a surprising result that we hypothesize is explained by their generally stronger metacognitive skills. We also report the most common difficulties described by students, which include challenges with syntax, error messages, and the process of compilation, and suggest teaching practices that might help students as they transition to a new programming language.

CCS CONCEPTS

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

KEYWORDS

language transition, programming, sentiment, metacognition

ACM Reference Format:
Paul Denny, Brett A. Becker, Nigel Bosch, James Prather, Brent Reeves, and Jacqueline Whalley. 2022. Novice Reflections During the Transition to a New Programming Language. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2022), March 3–5, 2022, Providence, RI, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3478431.3499314

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

ACM ISBN 978-1-4503-9070-5/22/03.
https://doi.org/10.1145/3478431.3499314

1 INTRODUCTION

Over the course of a typical undergraduate computing degree, students take a variety of courses in which they are expected to write programs in multiple different languages. Computing students therefore transition from one language to another several times during the course of their studies, each time building on the knowledge and concepts they have developed in earlier languages. Some students may view these transitions positively, as building competence in multiple languages contributes to their growing identity as computing professionals [30]. Other students may take a more negative view, finding the transitions difficult and bothersome, preferring to rely on their existing and familiar programming knowledge. Understanding how students perceive learning new languages is important, given the strong connections between emotion, self-efficacy and performance [15, 20, 25].

Although conventional wisdom states that learning a new programming language is easier after a first language is mastered [41], in practice such learning transfer may not be straightforward. When two languages use the same syntax for a particular concept, but have different semantics, this can interfere with learning [39]. Transitioning from a block-based to a text-based environment involves unique challenges [28], and demands appropriate support, compared to transitioning between procedural or object-oriented languages [29]. Explicit instruction supporting the learning of a new language can be effective, but requires a good understanding of the difficulties that students encounter for their particular language transition [17, 40]. Much of the prior work exploring these challenges has derived insights from programming artifacts or researcher-created instruments, rather than from the student voice.

In this work, we explore how students perceive the transition from an interpreted language (MATLAB) to a compiled language (C), and what difficulties they encounter. We seek to answer:

RQ1. What barriers do novice programmers face when beginning the transition to a new programming language?
RQ2. To what extent do novice programmers enjoy or dislike learning a new programming language, and how does this sentiment relate to their performance?

2 RELATED WORK

The process of learning an additional programming language has been a topic of interest in the computing education community for

decades. In 1970, researchers and teachers were already thinking about which language was best as a *first* language to make learning additional languages easier [23]. By 1990, it was well known that both novices and seasoned professionals can struggle with the transfer of knowledge from one language to another [35, 36]. Even moving from the procedural to the object-oriented version of the same language can be difficult [29]. Recent data-mining from resources like StackOverflow show that this is still a concern [37].

Transfer from block-based languages. A popular move in linguistic transfer of programming languages is to first use block-based languages like Alice or Scratch to teach concepts that can then be transferred. Students who first learn block-based programming often need less time to learn concepts in new languages [2]. However, there is evidence that this transfer should be explicitly taught. Powers et al. found students struggled to switch from Alice to a text-based language without support [31], whereas a specific pedagogy for the transfer developed by Dann et al. proved successful [8].

Yet block-based environments are not without their shortcomings. Moors et al. discuss their significant weaknesses as a way of transitioning into text-based languages, such as their focus on animation and multimedia that is typically lacking in their text-based counterparts [28]. Weintrop and Wilensky found no automatic benefit in starting with a block-based language and then transitioning to a text-based one [42]. They argue that there is a need for educators and tool designers to work to help facilitate this transition.

Tools to help mediate the transition. Researchers and teachers have been creating tools to help students move from one language to the next for decades. In 1996, Fix and Wiedenbeck presented ADAPT, a tool which helped students who already knew one language transition to Ada [13]. More recently, Hundhausen et al. found a positive transfer effect with their tool, ALVIS Live, in which users fill in dialog boxes and directly manipulate program data [19]. Holvitie et al. built ViLLE to assist students transitioning from Python to Java [17], reporting that student outcomes improved when it was used. Still others have shared their tools without empirical evidence, but report student experiences were positive [21, 22], or had inconclusive results [7]. Overall these reports seem to indicate that students who have learned a first language may benefit from some kind of assistance, either through a tool or from a teacher, when applying what they have learned to a new language.

Metacognition and learning a second programming language. Most studies to date have not been deeply theoretically motivated and those that have focus on theories such as mediated learning [8]. Very recent work has sought to understand this process from the perspective of natural language acquisition [39, 40]. In this paper, we utilize Winne & Hadwin's model of self-regulated learning, a theory of metacognition, as an explanatory model that makes sense of previous work as well as the results of our own study [43]. Recent work has shown an increased interest in utilizing metacognitive theory in computing education [32]. To our knowledge, this is the first study in computing education to utilize this particular metacognitive model.

Winne & Hadwin's model is useful here because it splits Bandura's [3] forethought phase into two parts: task definition and goal setting/planning [43]. The other two phases are enacting strategies and metacognitive adaptation (i.e. self-reflection). This model treats information from internal and external sources as a feedback loop between the current state and the goal state [32]. Each phase includes five components: conditions, operations, products, evaluations, and standards. Learning a new programming language fits into this model quite well and can explain much of the learning difficulties presented by prior research.

Previous work has shown that even experienced programmers need help in adapting their high-level plans (i.e. goal setting) to the new language [36]. Those who are only familiar with a single programming paradigm sometimes draw incorrect parallels (i.e. task setting) when trying to transfer what they know to a different paradigm, such as procedural to OO [29]. Even within the same paradigm, programmers make failed attempts to relate a new language with what they already know, mixing up syntax and concepts [37].

Walker and Schach found that students learning Ada as their second language primarily exhibited two types of undesirable behavior [41]. The first was using non-Ada constructs instead of learning how to use the features of Ada, i.e. students were programming in Ada as if it were the language with which they were already familiar. This would mean they weren't understanding the underlying constructs of the language and therefore could not take advantage of its unique features. In Winne & Hadwin's model, this would fall into a failure of enacting the correct strategies. The second undesirable behavior the researchers noted was that students would attempt to use Ada constructs, but then when those didn't work as expected the students would remove them and fall back on non-Ada-specific (i.e. generic) constructs. This would mean students identified the correct goal, enacted the correct strategies, but short-circuited the feedback loop with incorrect metacognitive adaptation when comparing their own evaluation to the standard.

We believe that a theory of metacognition is best equipped to explain why students struggle with learning a new programming language and to help us better understand these difficulties.

3 METHODS

In this study, we explore the written reflections of first-year engineering students at the University of Auckland, a large public research university in New Zealand. All engineering students at this institution take a compulsory introductory programming course which covers two languages, MATLAB and C, taught in consecutive 6-week modules. In 2020, when we conducted this study, 1,030 students were enrolled in the course. Students were invited to respond to an ungraded prompt during the first week of the second module, at which time they were switching to the C programming language. The data we present here is based on the reflections provided by 771 students who responded to this prompt.

3.1 Course context & data

3.1.1 MATLAB. The first module of the course introduces students to MATLAB, which offers a fully-featured integrated development environment. MATLAB is dynamically typed, such that the programmer does not explicitly define types for variables, and is an interpreted language as scripts are executed within the MATLAB environment without a compilation step. In the first module of the

course, students learn concepts typical of a CS1 course including variables, arithmetic, arrays (vectors), functions, flow of control and basic algorithms, and MATLAB-specific topics such as 2D and 3D plotting and standard matrix operations.

3.1.2 C. After completing the MATLAB programming module, and after a two-week mid-semester break, students are introduced to the C programming language. In comparison to MATLAB, C is statically typed as variable types are specified by the programmer and checked by the compiler prior to execution. In the later weeks of this module students are introduced to an integrated environment (Microsoft's Visual Studio). However during the first week of the transition – when students wrote their reflections for this study – they edited their source code using a text-editor and compiled using command-line tools.

3.1.3 Reflective prompt. During the first week of the C programming module, students were invited to respond to a prompt that asked them to reflect on their experience transitioning from MATLAB to C. The full wording of the prompt is shown in Figure 1. Responding to the prompt was an optional, ungraded task.

3.1.4 Performance data. Graded activities in the course include weekly labs, one programming project and one test for each module, and a final exam. As a result of on-campus restrictions during 2020, only two assessments were held in-person and invigilated (proctored). These two invigilated assessments were the test for the C module and the final exam. Our interest is understanding how student perceptions of the transition to the C language relate to their subsequent performance in the C module of the course. To measure performance in the C module, we combine the weekly lab scores, the programming project, and the invigilated test and exam scores for this module. The invigilated scores are weighted more highly, consistent with the grading approach used in the course.

> You have been learning MATLAB for 6 weeks now, and have just begun to learn the C programming language. You will notice some things are different, and some are similar. You may find it easier to learn another language, given that you already understand basic programming principles, or you may find it harder because some things work differently to what you are used to. Everyone will experience this differently.
>
> Write a short reflective piece (anywhere from several sentences to several paragraphs) in your own words, that describes how you have found this transition to learning a new language. What are you finding easier, and what are you finding harder about learning C compared to MATLAB?

Figure 1: Reflective prompt shown to students.

3.2 Thematic analysis

The student responses were analyzed using a grounded theory approach [14]. One author independently performed an open coding of 119 student responses until saturation was reached, at which point no new codes emerged from the data. Responses were examined in a random order during this initial phase. Two different authors then coded a small random subset (30) of these responses using the codebook produced by the initial author. The three authors then met to discuss the coding, and finalise the descriptors and examples associated with each code. Although our main interest, with respect to RQ1, was to understand the *difficulties* that students faced during the transition, students were prompted to comment on all aspects of the transition and our thematic analysis captured both positive and negative statements.

The final version of the codebook consisted of 22 codes, which were organised into four categories: 'difficulty', 'prior knowledge', 'language aspects' and 'tools/environment'. A total of nine codes captured responses about the relative difficulty of the MATLAB and C languages, and students' perceptions of the transition. Three codes were used to classify statements regarding the benefits and drawbacks of prior knowledge, and any languages explicitly mentioned for which students had prior experience were recorded. Six codes captured language aspects, including similarities and differences in syntax and conceptual understanding, as well as statements about the transition to a statically typed and compiled language. Finally, four codes were used to classify responses relating to setting up tools and changing to a new programming environment.

Once the codebook was finalised, all three authors independently coded 146 randomly selected student responses (representing 20% of the dataset). In all cases where one coder differed from the other two, they reviewed their decision to ensure it was not the result of a data entry error or trivial oversight. After elimination of trivial errors the raters achieved 84% unanimous agreement across all 535 codes assigned to the 146 statements, yielding a Bray–Curtis dissimilarity score ranging from 0.03 to 0.04 for all rater-pairs. Final consensus was then achieved when two or three raters agreed on a code, resulting in a total of 477 codes assigned across all 146 student responses. The most commonly occurring individual codes were identified, and an axial coding step involving a discussion between all three coders was used to make connections between codes and identify the primary themes which we report in Section 4.1.

3.3 Sentiment analysis

To measure the sentiment of the students' reflections, we used a deep learning model for sentiment analysis that is provided as part of the Stanford CoreNLP natural language processing library [26]. The provided model has been trained on the Stanford Sentiment Treebank, a corpus of fully labelled parse trees extracted from movie reviews and consisting of more than 200,000 unique phrases [38]. The code and documented examples of use are available on the Stanford NLP website[1].

The model takes, as input, a section of text and classifies each sentence in the text as having either a negative, neutral or positive tone. To illustrate, a three-sentence reflection provided by one of the students in our study was classified by the model as follows:

- Sentence 1: *"Similar things are the basics such as inputting variables and using different math functions to solve them"* => **Neutral**
- Sentence 2: *"However, the syntax is quite different in how you would write the code, so that may take a bit of getting used to I believe"* => **Negative**
- Sentence 3: *"Even though this has just started, I'm quite enjoying learning a new language and expanding my language pool, so I'm excited to see what else we will learn!"* => **Positive**

[1] https://nlp.stanford.edu/sentiment/code.html

For a given student, the sentiment analysis produces the number of sentences in the response they provided that had a positive, neutral and negative tone. The overall sentiment for a student was computed as the number of negative statements subtracted from the number of positive statements. In the example shown above, the student has expressed an overall sentiment of 0 (with one positive and one negative classified statement). Our hypothesis was that a positive relationship would exist between overall sentiment and subsequent performance. That is, students who provided reflections using more positively toned statements would be more likely to achieve higher scores in the C module.

3.4 Metacognitive analysis

An NLP method was used to measure metacognition in the student reflections, because sentiment alone has a complex relationship with learning. In some cases, negative sentiment can be positively related to learning especially where students experience challenges that they are able to overcome or *desirable difficulties* [5]. Conversely, difficulties that are not resolved may negatively impact learning [11].

The NLP method we applied [6, 18] detects metacognitive phrases that start with a first-person pronoun (e.g., I, me), end with a metacognitive indicator word (e.g., know, thought), and may have additional words between the two. For example, in Sentence 2 above, which was flagged by the sentiment analysis as negative, the metacognition detection method flagged the phrase *"I believe"* as metacognitive.

4 RESULTS & DISCUSSION

Our findings are organised in two sections. The first section, which addresses RQ1, presents the results of our thematic analysis involving approximately 20% of the reflection statements in our dataset. We primarily focus on the difficulties described, as these may suggest areas in which we can provide better support to students in the future. We also report the main themes that reflect positive feedback from the students. The second section, which addresses RQ2, explores the sentiment and use of metacognitive phrases in student responses and investigates how this relates to their subsequent performance in the course when using the new language.

Table 1: The five most frequent codes corresponding to difficulties faced by students transitioning from MATLAB to C.

%	Code	Descriptor (abridged)
31.5	Syntax_Differences	explicit mention of different syntax
24.7	Type_System	difficulty with typing / variable types
18.5	Interpreted/Compiled	manual compilation and execution
17.8	IDE/Environment	challenges related to new environment
13.0	Errors/Debugging	problematic error messages, debugging

4.1 Themes emerging from reflections

Recall that students were asked to describe their experience of the transition, and were explicitly prompted to state what they were finding easier and harder about learning the new language. Half of the students (51%) made some mention of the fact that having *prior knowledge of programming helped* in their transition. This was, in fact, the most frequently coded type of statement in the dataset, across all 22 codes that were used. For most students, their prior experience with MATLAB was implicit in this type of response, however around 15% of the responses in our sample explicitly listed other languages. Searching for any occurrence of languages listed in the full dataset of 771 responses, students indicated prior experience with Python (70), C++ (20), Scratch (10), and Javascript and HTML (both with 9 mentions).

We now present the main challenges that students wrote about in their reflections. Table 1 lists the five most frequent codes that correspond to difficulties encountered during the transition. The frequency of each code is given, alongside the descriptor (which is abridged for space reasons).

4.1.1 Syntax challenges. Differences in the syntax between MATLAB and C were mentioned by one-third of the students (31.5%), making it the most frequently cited challenge. Familiarity with syntax is developed through practice over time, and this may be a particular challenge in courses where students are expected to learn two languages during a single term. Statements representative of the challenge of transitioning to the new syntax include:

- *"The difficult part for me is getting used to the new C language and syntax."*
- *"It will take a bit of time to get used to adding semicolons at the end of every line, as well as some of the other little things that need to be added"*

Difficulty with syntax is a widely known challenge for novice programmers [1, 9, 34], and we find evidence here that it remains a major hurdle for students when learning a subsequent language. Deliberate practice of syntax may therefore be beneficial for some students, and has been the focus of recent work [24]. For example, Edwards et al. found that over a period of five weeks, as little as 25 minutes per week of syntax practice resulted in higher exam scores and lower course attrition rates [12].

4.1.2 Static typing and the compilation process. The next two most common difficulties were related to the transition to a compiled language, and in particular the static type checking that is performed by the C compiler. More than 40% of students mentioned difficulties relating to either types or the process of compilation. In contrast to MATLAB's dynamic typing, students found the need to declare variable types in C particularly problematic.

- *"The variable system feels overly complicated compared to MATLAB"*
- *"I think the hardest part is to identify and classify the type of variables. Compare with MATLAB, the type of variable create many unnecessary trouble"*

When writing their reflections, during the first week of learning C, the benefits of static typing may not have been clear to students given the relatively simple, and short, programs they were writing at the time. For larger software systems, there is compelling empirical evidence that static type checking improves maintainability and debugging time [16]. Indeed, the few students who had entered the course with prior experience using statically typed languages preferred static type checking and explicitly noted the benefits:

- *"I prefer C to MATLAB because of the static typing system that I am accustomed to from learning C++ and other statically typed languages. I disliked MATLABs approach as it can be harder to find bugs and MATLAB will generally try and make things work, often resulting in it doing something unintended."*
- *"I already knew C before taking the course, so it's been much more straightforward for me than MATLAB. I prefer the more structured nature of C, with explicit declarations and compile time checking for typos in variable names and type checking."*

Using a command-line compiler was also a challenge for many students. They noted the additional burden of having to manually compile their programs before execution.

- *"For C, needing to compile the code and then executing it in Developer Command Prompt is a lot more time consuming than being able to run the code straight away like MATLAB"*
- *"The thing I found most difficult was getting the hang of the command prompt in order to compile, as previously the software would automatically compile for me"*

4.1.3 Debugging and error messages. The difficulty of locating and fixing bugs was a common challenge. This appeared to be a combination of the cryptic and unhelpful messages produced by the C compiler, as well as observations that the MATLAB environment provided more accurate information regarding the location of errors.

- *"learning C is definitely more difficult as it is much harder to find errors as opposed to MATLAB which provides detailed information on where your code is wrong"*
- *"The only difficulty i've had is reading and understanding error messages"*

Again, there is much prior literature documenting the challenges faced by novices in understanding compiler error messages [4]. In our case, the problem may have been exacerbated by the more pleasant experience students had when working in the MATLAB environment. Certainly, locating an error in a source file using the feedback from a command line compiler is less straightforward than having the line automatically highlighted in the environment. Students may have reported fewer difficulties with error messages and debugging had the language transition been in the opposite direction, from C to MATLAB.

Providing direct instruction to help students become familiar with the new error messages may be helpful, and could be combined with the syntax practice activities recommended earlier. In situations where students are using online programming tools, another approach may be to adapt the presentation of error messages as familiarity develops – that is, present more understandable messages earlier, and remove this scaffolding over time. Such approaches to error message enhancement have proven effective in practice [10].

4.1.4 The benefits of prior knowledge. Despite the many challenges students reported in transitioning to C, and the fact that many more students stated learning C (32%) was harder than learning MATLAB (8.2%), in general they were overwhelmingly positive towards the benefits that prior programming knowledge offered in easing the transition. More than half of the student responses explicitly stated that having prior knowledge helped (51%), whereas very few stated that this prior knowledge hindered their learning of the new language (2.1%).

- *"I am finding C slightly easier (so far) compared to Matlab since I am more familiar with the key principles whereas for MATLAB everything was completely new"*
- *"I have found the transition relatively easy now understanding some basic programming principles."*

The prevalance of this code may have been influenced by the wording of the prompt (see Figure 1) which included both a positive and negative example of how prior knowledge may help or hinder. In addition, students were asked to write their reflections during the first week of learning C, at which point they had encountered only a small subset of the syntax and concepts taught in the 6-week module. Many of the initial topics – such as assignment to variables and arithmetic – use a similar syntax and thus may provide few conceptual challenges following the 6 week MATLAB module. Prior experience is more problematic when the same syntax has different semantics in the two languages – what Tshukudu et al. refer to as 'false carryover constructs' [39]. It is possible that students in our course may have had a different view of their prior knowledge once they encountered such constructs. An example is C's 0-based indexing, which differs from MATLAB's 1-based indexing, and is known to contribute to the high frequency of off-by-one errors encountered by novices in introductory courses [27, 33].

4.2 Sentiment, metacognition & performance

We expected that students who wrote more positively toned statements in response to the language transition prompt would be more likely to perform well in the subsequent course activities that used the new language. For the purpose of our analysis, we organise students into quartiles by defining score ranges that divide the students into four groups that are as evenly-sized as possible. Table 2 shows the score ranges for each quartile along with the length of the responses produced by all students in a quartile, and the results of the sentiment and metacognitive analyses.

Table 2: The total length (in characters) of written responses, the number of positive (Pos), neutral (Neu) and negative (Neg) statements and the number of metacognitive (MC) statements, organised by quartile (Q).

Q	*n*	Scores	Length	Pos	Neu	Neg	MC
*Q*1	191	93.0-100.0	106,396	188	242	337	299
*Q*2	190	86.4-92.9	87,737	185	211	269	249
*Q*3	192	75.2-86.3	82,794	176	236	272	246
*Q*4	198	0-75.1	73,102	160	202	210	259
Total	771	-	350,029	709	891	1088	1053

Higher performing students tended to write longer responses and were more inclined to write negatively toned sentences. For example, the average overall sentiment (computed as the difference between the number of positive and negative statements) for students in the highest performing quartile, Q1, is -0.78 compared to -0.25 for students in Q4. To better illustrate this, Figure 2 shows the

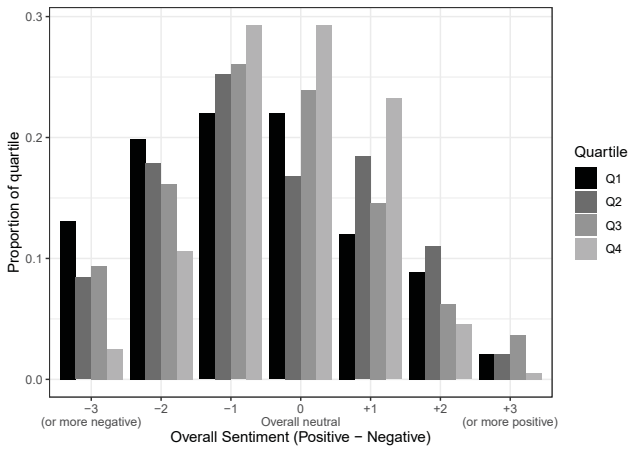

Figure 2: Overall sentiment of responses by quartile (number of positively toned minus number of negatively toned statements). Each bar denotes proportion of students in quartile giving a response with corresponding sentiment.

distribution of overall sentiment broken down by quartile (proportions are plotted, so for each quartile bar heights sum to 1). The leftmost set of four bars represents the students with the most negative overall sentiment (-3 or more), and these students were much more likely to perform well in the course (13% of Q1 are in this group compared to less than 3% of Q4). To interpret this finding, we consider that the ability to identify and verbalise difficulties and anticipate future challenges demonstrates higher self-reflective capability – metacognitive adaptation within the self-regulated learning model of Winne & Hadwin [43] – and is thus a positive predictor of future performance.

We find additional support for this interpretation as the highest performing students tended to write more metacognitive statements than the lowest performing students (299 statements for Q1 versus 259 for Q4; Table 2). However, the differences here are small, and it must be noted that the automated metacognitive classification is sensitive to the specific phrasing that students use.

In Section 4.1 we reported the main themes emerging from all student reflections. We observe an interesting trend when additionally taking into account subsequent performance in the C module. The second most frequent type of statement coded in our thematic analysis (36%) involved students indicating that they found the *transition straightforward, simple or 'OK'* (or other synonymous terms). The lower performing students in the course were much more likely to have made such a statement. Across the four performance quartiles, decreasing in performance from Q1 to Q4, the proportion of students who made a statement classified in this way was: 29%, 36%, 39%, 48%. This trend – a negative correlation between the perceived ease of the transition and subsequent performance – appears to mirror that seen in our automated sentiment analysis.

Finally, the representative quotes provided above fit well into the model by Winne & Hadwin. Students answering the prompt were specifically in the final metacognitive adaptation phase as they reflected on their transition from MATLAB to C. We can see examples of how their reflections fit into the COPES (Conditions, Operations, Products, Evaluations, Standards) acronym, which describes the different facets of each metacognitive phase. Students frequently mentioned the Conditions, which are both the reality of each language (i.e. syntax, typing, etc.) and their own personal prior experience with either language. Examples of Operations are found in student reflection on error messages and their difficulty understanding them. The Products include the code students had written to that point in either language, as well as information they've learned and their own cognitive offloading of that information (i.e. notes). Evaluations abound in the data as students compare their Products to both their own personal Standards as well as the standard for each language. We believe our analysis provides support for using a metacognitive model to understand language transition and look to future work for further investigation.

4.3 Threats to validity

We presented challenges described by students when reflecting on the very early stages of transitioning from MATLAB to C. We cannot generalise these findings to transitions between other languages or claim that reflections would be similar if collected at a later stage. Indeed, we find that some of the stated difficulties (e.g., those related to static typing) are specific to our context. Future work should explore learner reflections of transitions in other contexts.

In this study we lacked demographic data for the students including potentially relevant background knowledge such as programming experience prior to enrolment in the course. We did, however, observe that students seemed willing to share this information in their statements. It also seems likely that very few of the students had experience with C or any statically typed compiled language. Only 2.5% of students mentioned prior experience with C++, and in an earlier survey of incoming students to the university, roughly 2% indicated familiarity with C.

5 CONCLUSIONS

Students encounter several key challenges when transitioning from one programming language to another. Traditionally, prior work has analyzed program artifacts or used researcher-created instruments to understand these difficulties. In this paper, we asked students to reflect on the transition in their own words, collecting statements from 771 students transitioning from MATLAB to C. We found that adjusting to the syntax of the new language was the most common challenge, and that students also struggled with the transition to static type checking as well as finding errors and understanding cryptic error messages. Contrary to our expectations, we found that students who used more negatively toned statements in their reflections were more likely to perform well when using the new language. This may be a consequence of lower performing students not recognising where they are struggling, or where they are likely to struggle in the future, due to less well-developed metacognitive skills. As educators, we can help students transition to a new language through syntax drilling activities, explicit highlighting of the differences between languages, teaching error messages, and utilizing reflective activities that strengthen metacognition.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Shahil Lal, and Arto Hellas. 2018. *Learning Programming, Syntax Errors and Institution-Specific Factors.* ACM, NY, NY, USA, 90–96. https://doi.org/10.1145/3160489.3160490

[2] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From Scratch to "Real" Programming. *ACM Trans. Comput. Educ.* 14, 4, Article 25 (feb 2015), 15 pages. https://doi.org/10.1145/2677087

[3] Albert Bandura. 1986. The Explanatory and Predictive Scope of Self-Efficacy Theory. *Journal of social and clinical psychology* 4, 3 (1986), 359–373.

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. In *Proc. of ITiCSE WG Reports (ITiCSE-WGR '19).* ACM, NY, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[5] Elizabeth L. Bjork and Robert A. Bjork. 2011. Making Things Hard on Yourself, but in a Good Way: Creating Desirable Difficulties to Enhance Learning. In *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society*, Morton Ann Gernsbacher, Richard W. Pew, Leaetta M. Hough, and James R. Pomerantz (Eds.). Worth, NY, NY.

[6] Nigel Bosch, Yingbin Zhang, Luc Paquette, Ryan S. Baker, Jaclyn Ocumpaugh, and Gautam Biswas. 2021. Students' Verbalized Metacognition During Computerized Learning. In *Proc. of CHI 2021.* ACM, NY, NY, 680:1–680:12. https://doi.org/10.1145/3411764.3445809

[7] Neil Brown, Charalampos Kyfonidis, Pierre Weill-Tessier, Brett Becker, Joe Dillane, and Michael Kölling. 2021. A Frame of Mind: Frame-Based vs. Text-Based Editing. In *Proc. of UKICER 2021.* (Glasgow, UK) *(UKICER '21).* ACM, NY, NY, USA, Article 2, 7 pages. https://doi.org/10.1145/3481282.3481286

[8] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated Transfer: Alice 3 to Java. In *Proc. of SIGCSE 2012 (SIGCSE '12).* ACM, NY, NY, USA, 141–146. https://doi.org/10.1145/2157136.2157180

[9] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. Understanding the Syntax Barrier for Novices. In *Proc. of ITiCSE 2011 (ITiCSE '11).* ACM, NY, NY, USA, 208–212. https://doi.org/10.1145/1999747.1999807

[10] Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proc. of ITiCSE 2020* (Trondheim, Norway) *(ITiCSE '20).* ACM, NY, USA, 480–486. https://doi.org/10.1145/3341525.3387384

[11] Sidney K. D'Mello, Blair Lehman, Reinhard Pekrun, and Art Graesser. 2014. Confusion Can Be Beneficial for Learning. *Learning and Instruction* 29, 1 (2014), 153–170. http://www.sciencedirect.com/science/article/pii/S0959475212000357

[12] John Edwards, Joseph Ditton, Dragan Trninic, Hillary Swanson, Shelsey Sullivan, and Chad Mano. 2020. Syntax Exercises in CS1. In *Proc. of ICER 2020 (ICER '20).* ACM, NY, NY, USA, 216–226. https://doi.org/10.1145/3372782.3406259

[13] Vikki Fix and Susan Wiedenbeck. 1996. An Intelligent Tool to Aid Students in Learning Second and Subsequent Programming Languages. *Computers & Education* 27, 2 (1996), 71–83. https://doi.org/10.1016/0360-1315(96)00022-X

[14] Barney G. Glaser and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine de Gruyter, NY, NY.

[15] Patricia Haden, Dale Parsons, Krissi Wood, and Joy Gasson. 2017. Student Affect in CS1: Insights from an Easy Data Collection Tool. In *Proc. of Koli Calling 2017.* ACM, NY, NY, USA, 40–49. https://doi.org/10.1145/3141880.3141881

[16] Stefan Hanenberg, Sebastian Kleinschmager, Romain Robbes, Éric Tanter, and Andreas Stefik. 2014. An Empirical Study on the Impact of Static Typing on Software Maintainability. *Empirical Software Engineering* 19, 5 (2014), 1335–1382. https://doi.org/10.1007/s10664-013-9289-1

[17] Johannes Holvitie, Teemu Rajala, Riku Haavisto, Erkki Kaila, Mikko-Jussi Laakso, and Tapio Salakoski. 2012. Breaking the Programming Language Barrier: Using Program Visualizations to Transfer Programming Knowledge in One Programming Language to Another. In *2012 IEEE 12th Int. Conf. on Adv. Learning Tech.* (Rome, Italy). IEEE, NY, USA, 116–120. https://doi.org/10.1109/ICALT.2012.186

[18] Eddie Huang, Hannah Valdiviejas, and Nigel Bosch. 2019. I'm Sure! Automatic Detection of Metacognition in Online Course Discussion Forums. In *Proceedings of the 8th Int. Conf. on Affective Computing and Intelligent Interaction (ACII 2019).* IEEE, Piscataway, NJ, 241–247. https://doi.org/10.1109/ACII.2019.8925506

[19] Christopher D. Hundhausen, Sean F. Farley, and Jonathan L. Brown. 2009. Can Direct Manipulation Lower the Barriers to Computer Programming and Promote Transfer of Training? An Experimental Study. *ACM Trans. Comput.-Hum. Interact.* 16, 3, Article 13 (Sept. 2009), 40 pages. https://doi.org/10.1145/1592440.1592442

[20] Päivi Kinnunen and Beth Simon. 2012. My Program is OK – Am I? Computing Freshmen's Experiences of Doing Programming Assignments. *Computer Science Education* 22, 1 (2012), 1–28. https://doi.org/10.1080/08993408.2012.655091

[21] Michael Kölling, Neil C. C. Brown, and Amjad Altadmri. 2015. Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. In *Proc of WiPSCE 2015* (London, UK) *(WiPSCE '15).* ACM, NY, NY, USA, 29–38. https://doi.org/10.1145/2818314.2818331

[22] D. Krpan, S. Mladenović, and G. Zaharija. 2017. Mediated Transfer From Visual to High-Level Programming Language. In *2017 40th Int. Conv. on Inf. and Comm. Technology, Electronics and Microelectronics (MIPRO)* (Opatija, Croatia). IEEE, NY, NY, USA, 800–805. https://doi.org/10.23919/MIPRO.2017.7973531

[23] MP Lee, JD Pryce, and A Harrison. 1970. Prolog as a First Programming Language. *WIT Transactions on Information and Communication Technologies* 7 (1970).

[24] Antti Leinonen, Henrik Nygren, Nea Pirttinen, Arto Hellas, and Juho Leinonen. 2019. Exploring the Applicability of Simple Syntax Writing Practice for Learning Programming. In *Proc. of SIGCSE 2019* (Minneapolis, MN, USA) *(SIGCSE '19).* ACM, NY, NY, USA, 84–90. https://doi.org/10.1145/3287324.3287378

[25] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. 2016. Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. In *Proc. of ICER 2016* (Melbourne, Australia) *(ICER '16).* ACM, NY, USA, 211–220. https://doi.org/10.1145/2960310.2960329

[26] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations.* 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

[27] Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: A Review of the Literature From an Educational Perspective. *Computer Science Education* 18, 2 (2008), 67–92. https://doi.org/10.1080/08993400802114581

[28] Luke Moors, Andrew Luxton-Reilly, and Paul Denny. 2018. Transitioning from Block-Based to Text-Based Programming Languages. In *2018 Int. Conf. on Learning and Teaching in Computing and Engineering (LaTICE)* (Auckland, New Zealand). IEEE, NY, NY, USA, 57–64. https://doi.org/10.1109/LaTICE.2018.000-5

[29] H.James Nelson, Gretchen Irwin, and David E. Monarchi. 1997. Journeys up the Mountain: Different Paths to Learning Object-Oriented Programming. *Accounting, Management and Information Technologies* 7, 2 (1997), 53–85. https://doi.org/10.1016/S0959-8022(96)00024-0

[30] Anne-Kathrin Peters and Arnold Pears. 2013. Engagement in Computer Science and IT – What! A Matter of Identity?. In *2013 Learning and Teaching in Computing and Engineering.* IEEE, NY, USA, 114–121. https://doi.org/10.1109/LaTiCE.2013.42

[31] Kris Powers, Stacey Ecott, and Leanne M. Hirshfield. 2007. Through the Looking Glass: Teaching CS0 with Alice. *SIGCSE Bull.* 39, 1 (March 2007), 213–217. https://doi.org/10.1145/1227504.1227386

[32] James Prather, Brett A Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. In *Proc. of ICER 2020.* 2–13.

[33] Liam Rigby, Paul Denny, and Andrew Luxton-Reilly. 2020. A Miss is as Good as a Mile: Off-By-One Errors and Arrays in an Introductory Programming Course. In *Proceedings of the Twenty-Second Australasian Computing Education Conference.* ACM, NY, NY, USA, 31–38. https://doi.org/10.1145/3373165.3373169

[34] Anthony Robins, Patricia Haden, and Sandy Garner. 2006. Problem Distributions in a CS1 Course. In *Proceedings of the 8th Australasian Conf. on Comp Ed - Vol. 52* (Hobart, Australia) *(ACE '06).* Australian Computer Society, Inc., AUS, 165–173.

[35] Jean Scholtz and Susan Wiedenbeck. 1990. Learning Second and Subsequent Programming Languages: A Problem of Transfer. *Int. J. Human–Computer Interaction* 2, 1 (1990), 51–72. https://doi.org/10.1080/10447319009525970

[36] Jean Scholtz and Susan Wiedenbeck. 1992. Learning New Programming Languages: An Analysis of the Process and Problems Encountered. *Behaviour & Info. Tech.* 11, 4 (1992), 199–215. https://doi.org/10.1080/01449299208924339

[37] Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. 2020. Here We Go Again: Why Is It Difficult for Developers to Learn Another Programming Language?. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)* (Seoul, Korea). IEEE, NY, NY, USA, 691–701.

[38] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proc. of the 2013 Conf. on Empirical Methods in NLP.* Association for Computational Linguistics, Seattle, Washington, USA, 1631–1642. https://aclanthology.org/D13-1170

[39] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proc. of ICER 2020* (Virtual Event, New Zealand) *(ICER '20).* ACM, NY, NY, USA, 227–237. https://doi.org/10.1145/3372782.3406270

[40] Ethel Tshukudu and Siri Annethe Moe Jensen. 2020. The Role of Explicit Instruction on Students Learning Their Second Programming Language. In *Proc. of UKICER 2020* (Glasgow, UK) *(UKICER '20).* ACM, NY, NY, USA, 10–16. https://doi.org/10.1145/3416465.3416475

[41] Karen P. Walker and Stephen R. Schach. 1996. Obstacles to Learning a Second Programming Language: An Empirical Study. *Computer Science Education* 7, 1 (1996), 1–20. https://doi.org/10.1080/0899340960070101

[42] David Weintrop and Uri Wilensky. 2019. Transitioning From Introductory Block-Based and Text-Based Environments to Professional Programming Languages in High School Computer Science Classrooms. *Computers & Education* 142 (2019), 103646. https://doi.org/10.1016/j.compedu.2019.103646

[43] Philip H Winne and Allyson F Hadwin. 1998. Studying as Self-Regulated Engagement in Learning. *Metacognition in educational theory and practice* (1998), 277–304.