Start coding in every human language?

Ahila Ramesh Rajamani[1], Narges Osmani[2], Ye Jin (Jinny) Kim[1], Janaya Paul[3], Christopher William Schankula[1], Juthada (Jamie) Malakulang[2] and Christopher Kumar Anand[1,2]

[1]*McMaster University, 1280 Main St W, Hamilton, ON L8S 4L8, Canada*
[2]*STaBL Foundation, 69 Wolfe St, Louisbourg, NS B1C 2J7, Canada*
[3]*Kji-Wikuom Studios, 15 Medicine Trail Rd, Eskasoni, NS B1T 1K5, Canada*

Abstract

Our outreach organization has introduced over 30,000 Grade 4 to 8 students to coding over the last decade. Using a process of iterative refinement, we have developed our own tools which work very well for introducing beginners to text-based coding for the first time. In our system, students start with ShapeCreator, an organized list of all the functions needed to create basic vector graphics, in which composing the functions is represented as a coloured ribbon threading through the function choices. However, because the coding language borrows words from English, there are many students who are not well-served by this system, including students who do not know English, and minority language students whose communities are trying to preserve their language and culture. In first case, young children, especially economically disadvantaged children, may not have been exposed to English, and this creates an additional barrier to learning. In the second case, minority language communities around the world trying to preserve their language and culture have set up immersion schools, but lack STEM material in their languages. This paper outlines the arguments available in the academic literature for teaching in children's native language and using code switching (intermingling languages) in bilingual populations. It demonstrates the technical feasibility of creating a Multi-Lingual ShapeCreator, and social viability of building a distributed team bringing together people from organizations embedded in Indigenous, cultural and educational communities. Finally, it outlines planned future work to extend code switching to advanced programming using a structure editor, and the communities preparing to test ShapeCreator in multiple languages around the world.

Keywords

computer science education, introduction to programming, native language education, code switching, computational thinking

1. Introduction

To open up career pathways and prepare students for a rapidly evolving world, it is vital that all children have a way into the world of software. The English-centric nature of programming languages is a barrier to many of those children. Although significant block-based languages already support language translation, we will argue that the literature already supports the extension of such support to text-based languages, and present a roadmap for providing it through structure editors for text-based languages. McMaster Start Coding (http://outreach.mcmaster.ca), a student service club, has introduced 30,000 children from grades 4 to 8 to computer science over the last decade. To support national and international expansion we established an associated charity, STaBL Foundation (https://stablfoundation.org). The charity is mandated to help all children, with a focus on children facing barriers, including Indigenous children, refugees, and children around the world with low access to technology. One way to increase access is to provide tools in multiple languages. Translation of the programming language is already available in block coding (e.g., Scratch [1] and Blocky), and we will

STEM@ICon-MaSTEd 2025: 4th Yurii Ramskyi STE(A)M Workshop co-located with XVII International Conference on Mathematics, Science and Technology Education, May 14, 2025, Ternopil, Ukraine
rameshra@mcmaster.ca (A. R. Rajamani); narges.osmani@stablfoundation.org (N. Osmani); kim630@mcmaster.ca (Y. J. (. Kim); janaya@kji-wikuomstudios.com (J. Paul); schankuc@mcmaster.ca (C. W. Schankula); jamie.malakulang@stablfoundation.org (J. (. Malakulang); anandc@mcmaster.ca (C. K. Anand)
https://www.cas.mcmaster.ca/~anand (C. K. Anand)
0000-0001-5102-4908 (N. Osmani); 0009-0004-1111-1029 (Y. J. (. Kim); 0000-0002-3721-549X (J. (. Malakulang); 0000-0002-7863-8595 (C. K. Anand)

Figure 1: Our outreach program mixes coding and storytelling, including frameworks for producing adventure games and animated comics, as shown on the left. Although children can write speech bubbles and other text in any language supported by unicode, up to now, the programming language remained based on English. This paper describes our plan and progress on translating the language and tooling into other human languages.

see that researchers have measured a positive impact of providing support in learners' native languages. We are not aware of other efforts to extend such support to text-based languages. Text-based languages are generally supported by tools developed independently, including text editors, version control and compilers. Coordinating translation across this ecosystem would be difficult. So our approach will be to use a structure editor built into our Web-based Integrated Development Environment (WebIDE).

Given the lack of specialist computer science (and even math) teachers at the primary level, our tools are designed to require little background knowledge beyond basic geometry and number sense. Conversely, children who follow our Algebraic Thinking curriculum [2] will be better prepared for high school algebra, since that has been identified as a barrier to higher education pathways.

In section 2, we outline the evidence in support of code switching (making use of multiple languages in education) and native language support specifically for teaching programming. Then, in section 3, we describe our existing Algebraic Thinking curriculum, how it influenced the design of our graphics library and the presentation of that library in our exploration tool ShapeCreator. Next, in section 4, we explain how our new ShapeCreator translates code into multiple languages. This is followed by section 5 describing how we plan to extend our approach to general programming using a structure editor, our priorities on curriculum translation, and plans for evaluation. Finally, we present recommendations for educators in section 6.

1.1. Contribution

Our contribution in this paper is the demonstration of support for multiple human languages in a text-based programming language. Current support extends to our first teaching tool, ShapeCreator. Technically, this mechanism for translating the tool relies on the fact that it is a structure editor. Any language supported by a structure editor could be translated with the same ease.

2. Related work

The body of literature surrounding languages and introduction to programming is extensive. Without doing a systematic review, we capture some highlights about several aspects: (1) the role of algebra and programming as languages in learning, (2) the negative impact on non-English speakers of English-centrism, and (3) the advantage of code switching, generally and in the programming context. Finally, we introduce the history of structure editors, since we will later argue that they provide a pathway to supporting learning in multiple languages.

2.1. Algebra and programming through a language lens

Algebra is a mandated course in most countries, and is recognized as a gateway or gatekeeper course, because failure to succeed in algebra closes many educational and career pathways [3]. However, poor performance persisted, prompting Silver [4] to advocate for "algebra for all" including the integration of algebra and pre-algebra activities in diverse subjects before students formally see algebra in high school. This led to the formulation of Algebraic Thinking, in early grades by Kieran [5] and middle grades by Driscoll [6] and many others.

Adding a student's voice to the debate, Solomon [7] highlights the parallels between human languages and programming languages and how these parallels can be used to introduce programming languages to even young students. He also draws on knowledge about the way infants acquire language to propose a sequencing of concepts in early lessons, and criticizes the use of commercial languages, such as Java, in early instruction because of the overwhelming syntax required to do simple things.

Krishnamurthi and Fisler [8] add the perspective of current programming language research, and argue that different language families should be distinguished not by their surface features, but by the type of abstract machine used to define the semantics of the language. Not surprisingly, functional languages whose underlying sematics are defined via the evaluation of algebraic expressions, are used by the authors within the Bootstrap project [9] to connect programming and high school algebra.

2.2. Impact of non-native language

Hong et al. [10] remarked that since beginners are well-known to have many misconceptions about programming, having to translate keywords from English to their native language adds another space where misconceptions can grow. This is reflected in the survey work of Alaofi and Russell [11] who studied 160 undergraduate students in two countries, and found widespread anxiety related to a working in a foreign language, with an overall significant negative impact on their assignment marks. Subsequently, they found through student interviews that even in undergraduate courses where instruction and testing was in English, 20% of students still reported difficulty understanding keywords, tooltips and other uses of English in the IDE [12].

The Scratch platform is used by millions of students, and has fostered collaborations between countries and continents. It does have non-English support, so it is natural to ask whether this support has had an impact. Dasgupta and Hill [1] correlated the rate at which Scratch learners added to their repertoire of blocks (abstract syntax nodes appearing in their program trees) to multiple variables, including whether they were using the English or translated interfaces. They found a 5% to 10% increase in repertoire when the interface was translated into the local language for their country. Feijoo-Garcia et al. [13] studied English-Second Language (ESL) students learning Scratch. The found that "the majority of ESL Learners (i.e. Hispanic participants) thought that they would perform better in their native language. This feeling of inadequacy can impact confidence in learning[...]" For example, Hispanic students took 5.65 minutes on average to familiarize themselves with the interface, which only took native English speakers 3.52 minutes. Students would definitely pick up on this difference, which explains their feeling of inadequacy. It would be interesting to try to duplicate these results in a classroom in a non-English-speaking country.

Soosai Raj et al. [14] studied two groups of undergraduate students in Tamil Nadu, learning about linked lists. One group was taught in English exclusively, and the other was taught using English/Tamil code switching, and found 96% of the code-switching grouped preferred that both languages be used, which even 67% of the control group would have preferred to switch. "More than 90% of the questions that were asked by the students in the experimental group during the lecture were in Tamil." Typical feedback (as quoted in the original paper): "The lecture was really very useful and it was easy to understand since the mixture of English and Tamil language helps us to learn better."

Through a thematic analysis applied to semi-structured teacher interviews, Dodoo et al. [15] found that second-language issues affected both students and teachers in US schools: "Students have to grasp the underlying concepts of programming and translate them into English while coding. This

dual cognitive load can be overwhelming, hindering students' ability to fully engage with and learn computing concepts." Teachers are well aware of these issues, and have found work-arounds, even teachers without knowledge of the non-English language (Spanish, in this case). For example, they do more hands-on coding, allowing students to observe them, and rely on visual aids.

Khan and Nabi [16] also explained issues with the non-native use of English using cognitive load: "In [Higher Education Institution]s the medium of instruction, [for various reasons], is officially English. This does not take into account the fact that students coming from Urdu medium background, and even many from English medium, are not sufficiently well versed in the English language. Thus the medium of instruction (English) can become extraneous cognitive load for these students and adversely affects their learning." An addition insight from their interviews is that the preference for language of communication is not the same for verbal and written communication. It is more important that verbal communication be in their native language. This is consistent with previous comments about students needing to translate concepts into their native language, because verbal communication needs to be translated simultaneously, but written communication does not.

Guo [17] analyzed 840 responses from programmers spanning 86 countries and 74 native languages, identifying several barriers faced by non-native English speakers. Their findings reinforce many of the ideas identified by other researchers including:

- "Respondents wanted instructional materials to have more visual imagery and multimedia (e.g., videos) rather than plain text, presumably because visuals can more easily transcend languages."
- "Non-native speakers must often rely on English-language instructional materials, some reported trouble with learning enough English to comprehend those materials while simultaneously learning the given programming concepts."
- "Non-native English speakers also reported trouble understanding source code. A common root cause is that programming language keywords (such as 'while') are in English."

We have focussed on the findings most relevant to primary and secondary education, but there are many more interesting findings in this thorough survey of learners.

Finally, Becker [18] supports many of the arguments above, and adds the observation that error messages in English are especially difficult for non-native English speakers.

2.3. Code switching

At one time, switching language while teaching and learning was actively discouraged or even forbidden [19]. In immersion programs, code switching would take away from time spent on the target language, and stop students from cutting the umbilical cord linking their new language knowledge to their native language. In many cases, however, this was also linked to class- and race-based discrimination, and in colonial settings to cultural genocide.

Today, code switching is recognized as a tool which can be used positively in and out of the classroom. It is one tool among many to aid in communication. In addition to human languages, programming languages and mathematics are now recognized as tools of communication, which can be studied in the same framework, with cross-fertilization of ideas. Even in the case of block languages used to program robots in kindergarten, Berciano et al. [20] found that translating from one language to another revealed issues with trainee-teacher understanding, and could be used for improving teacher education.

Contrasting different approaches to bilingualism (with and without code switching), Bers [21] comments that in a context of code switching "both languages interact to develop metalinguistic awareness, abstraction and higher forms of knowledge." In the context of learning programming languages, Hong et al. [10] found that learning both Python and C at the same time did not slow learning compared to a group learning only one language, as might be expected, given the different syntaxes involved. Spanning natural, programming, and coding languages, Schanzer et al. [9] have shown in multiple studies that translating between programs, symbolic algebra and English (in the form of word problems) improves learning and retention.

Finally, Ruby and Krsmanovic [22] survey the literature on what is known about teaching programming in English versus students' native language. Although they conclude that English knowledge is eventually required to reach advanced levels in the field, because of the extensive literature in English, it is likely that native language support would be beneficial for beginners. However, research is required to establish this, and there isn't even an accepted student-centred model for programming-language acquisition to parallel the ones for the acquisition of human languages.

2.4. Structure editors

Structure editors, called projectional editors by some authors, are editors which enable editing of the program structure (i.e, the Abstract Syntax Tree (AST)). Early structure editors [23, 24] enforced strict adherence to AST structures by template-based edit actions, requiring users to construct programs through drag-and-drop interactions. While these systems ensured precision and syntactic correctness, they were rigid and unintuitive for users accustomed to free-form text editing. Subsequent developments introduced more adaptive systems supporting free-form editing, where users input plain text code that is parsed into an AST within the editor [25, 26]. Modern structure editors further enhance usability by enabling seamless code entry and manipulation, mimicking traditional text editors while internally preserving AST-based constraints. Omar et al. [27], for example, laid a theoretical foundation for work in this direction by defining a typed structure-editor calculus.

3. Background

The student service club is aprt of an outreach program spanning two decades. Since 2016, our most common activity has been introducing children to coding in the Elm programming language, with a focus on a graphics library for creating vector graphics. The design of the library supports Algebraic Thinking. See [2] for a description of the graphics library and the philosophy of providing a small number of orthogonal functions through which complex graphics can be built using function composition.

Elm is a functional programming language. Today many programming languages like Swift, Kotlin, Scala, and Python are adopting functional programming features, but this introduces extra expressivity at the cost of increased complexity. Elm is a deliberately small language, and this support our educational purpose well, because there is less to learn, and it made it easy for the developers to create relatively simple and often prescriptive error messages.

We used the simple syntax and the focus on functions to create a composable graphics library in which graphics are created using an orthogonal set of functions with the minimum number of arguments possible. For example, `circle : Float -> Stencil` has one argument, which every child knows must be the radius if they have already learned about circles. The need for documentation is reduced, but not eliminated.

3.1. ShapeCreator

From the core language, children initially only need to understand list notation and "forward pipe" (|>) for composing functions. Most of their learning involves the functions in our graphics library, which are mostly meaningful to them from geometry. The challenge then is not of understanding, but of remembering which concepts from geometry are encoded available in our library. For this we created ShapeCreator as a single-page reference, which we later made interactive for use as a discovery tool, and finally added multiple-shape editing, so that initial tasks could be performed on the same page as the documentation.

Becker [18] observed error messages are already hard for beginners to understand, and when they are in English, it becomes even tougher for non-native English speakers to interpret and resolve them. This is obviously an argument for translating *all* aspects of tooling and documentation, but it is also an argument for using a structure editor (e.g., block coding) which eliminates errors entirely, and thus the need to translate them. This is one of the advantages with ShapeCreator (figure 2).

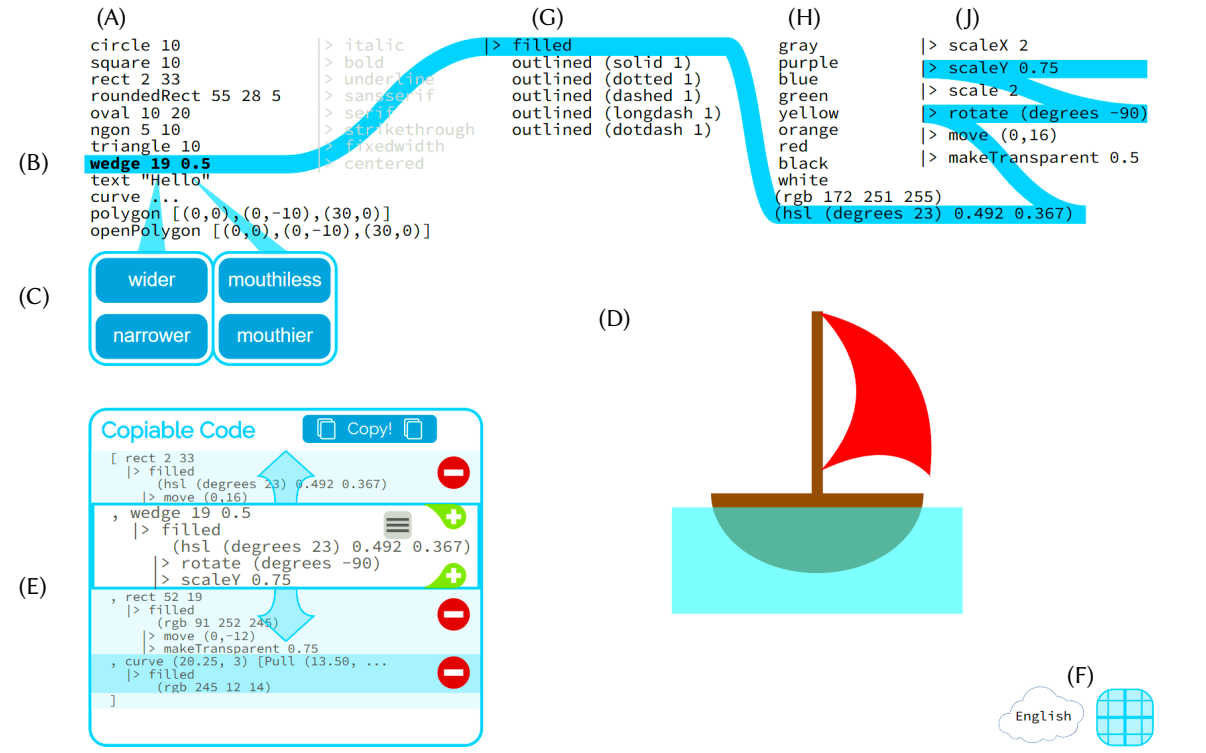

Figure 2: The ShapeCreator in English mode, showing (A) the list of stencil functions, (B) the start of the blue "ribbon" which shows how the functions are composed to create the graphic, (C) adjustment buttons for the currently selected stencil (i.e., the "wedge"), (D) the resulting composition of shapes, and (E) the code creating the composite shape. Utility buttons (F) allow change of language "graph paper" activation. Coming back to the ribbon. It follows the composition of functions in operational order, starting with the stencil, which is filled or outlined at (G) with a colour (H) to create a shape. Finally, zero, one, or many transformations can be composed at (J), which is indicated by a zig-zag in the ribbon.

The structure of ShapeCreator reflects the compositional structure of the library. In the first column (A) are the functions which create Stencil s. In column (G) are the functions which take a Stencil and fill or outline it in a colour (H), producing a Shape. Finally, transformations in column (J) can be applied to the resulting Shape. The blue ribbon starting at (B) emphasizes the composition of functions to produce the final shape.

In order to function as a structure editor for first coding tasks, a list (E) of Shapes can be selected for editing via the ribbon, or rearranged using touch or mouse actions. The resulting composite Shape is displayed at (D).

Teacher training is supported by videos on our STaBL Foundation YouTube Channel.

3.2. Activities

After being introduced to coding with ShapeCreator, children proceed to one of several activites which introduce collaboration, and meaningful goals. Currently, these activities require switching to an Integrated Web-Based Development Environment (WebIDE) (figure 3).

3.2.1. Wordathon

In the simplest activity, the teacher distributes reading words to the children, and they singly or in pairs code a vector graphic or animation of that word. Mentors then integrate the words into a reading game, using the project support built into the WebIDE (figure 3).

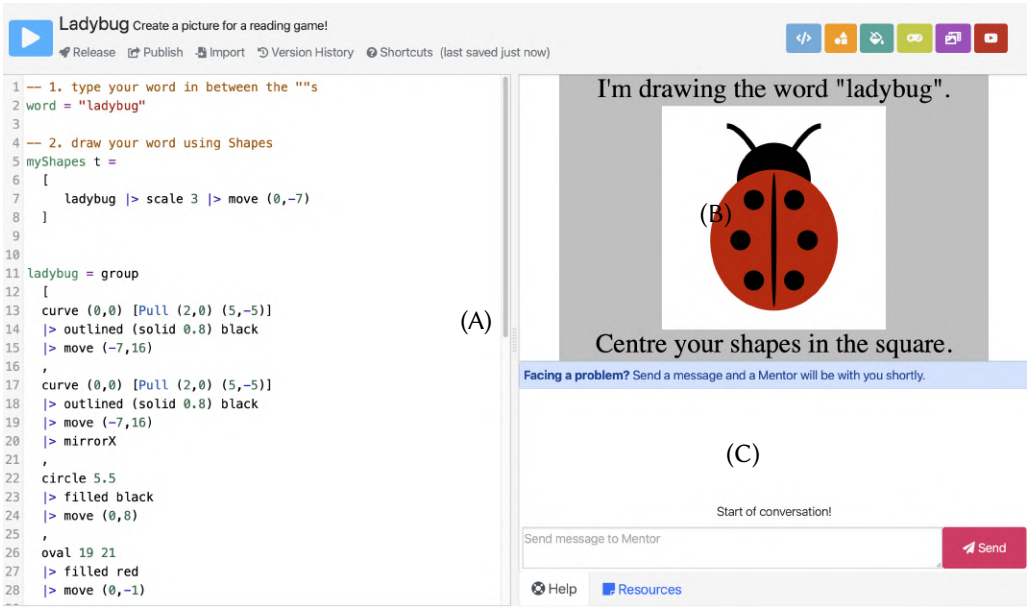

Figure 3: Web IDE showing a Wordathon Activity. In this case the child selected or was assigned "ladybug" to create in code (A), which is displayed in pane (B). Note the help panel (C) where children can ask mentors for help related to this code.

3.2.2. Herogram

Herograms are digital postcards in which children code an illustration or animation for the front, and write a message and address for the back. They can even modify the graphics for the stamp. Teachers can use this activity to celebrate holidays such as Mother's Day, or ask children to create a Herogram to thank a famous scientist, frontline worker, etc.

3.2.3. Comics

The comic activity introduces narration and calls for greater teamwork than in the Wordathon activity, because groups of children must agree on a story, characters, frame-by-frame breakdown of the story, and then assign tasks to produce the backgrounds, characters and text required. In figure 1, the finished product is displayed. The framework takes the individual frames and composes them into a traditional comic-book format, using multiple "pages" if necessary. The framework animates the enlargement of frames as they are clicked on. In this case the last frame of the page is highlighted. Children can also create animations within the frames. In this science-literacy example, the exchange of CO_2 and O_2 gases is animated.

3.2.4. Adventure Game

Narrative and teamwork are further developed in the Adventure Game. In this activity children draw a state diagram which represents the "map" of their adventure. States represent places, and transitions are rendered as buttons, which when clicked take the reader to the connected state. See [2] for an explanation of user interaction via state diagrams, and examples of how the web tool SDDraw is used to draw the state diagrams and generate a working adventure game, to which children add graphics and animation. Recently we tested a prototype version of SDDraw which allows multiple children to create the state diagram collaboratively. Twenty children and mentors recently created a New Year's adventure game[1] during a one-week hackathon (coding collaboration).

[1] https://stablfoundation.org/NewYearsMystery2025.html

3.3. ElmSr: a structure editor

Structure editors represent an alternative approach to programming environments by structuring code directly around the Abstract Syntax Tree (AST) rather than relying on plain text. This approach ensures that syntax and type errors are eliminated at the source, offering a robust and error-free programming experience. Tools like Scratch have demonstrated the potential of structure editors in educational contexts, particularly for teaching programming concepts to novices. However, text-based structure editors have faced resistance, often attributed to the perceived rigidity and unnaturalness of directly manipulating ASTs. Bridging this gap requires innovations that integrate intuitive user interactions with the underlying structural constraints of AST-based programming.

ShapeCreator is itself a structure editor, but it only supports a limited subset of the language and graphics library. ElmSr [28] is our project to support all of the language with extensible support for additional libraries, in a way which meets user expectations based on their experience with text editors, addressing the weaknesses of structure editors.

Unlike traditional text editors, structure editors enforce strict constraints on user input to ensure syntactic and semantic correctness. While this provides significant advantages in error prevention, the inflexibility of many structure editors has been a barrier to adoption. For structure editors to gain broader acceptance, they must balance the benefits of AST-based editing with interaction paradigms that align with user expectations shaped by conventional text-based systems.

To explore potential solutions to these challenges, we have developed an experimental structure editor, ElmSr, for the Elm programming language. ElmSr focuses on "natural" navigation and editing behaviours, aligning cursor movement and user actions with the visual representation of code rather than the underlying AST structure. This keyboard-driven editor enhances the user experience while maintaining the structural guarantees of AST-based editing. It supports intuitive navigation, provides a consistent experience for arithmetic expressions and includes a real-time, colour-coded feedback system for identifier validity, aiding error detection and correction. Additionally, ElmSr was developed to support our Algebraic Thinking curriculum, incorporating shortcuts for common tasks to help users efficiently progress from simple to complex programs. While there are many reasons researchers want to make structure editors widely adoptable, for this paper, the key advantage is that a structure editor renders the text based on the AST, so it is easy to make rendering match the user's language preference.

Figure 4 shows ElmSr's graphical interface, which is divided into functional areas. The left pane is the code editor, where users can enter and edit code, with cursor location highlighted in yellow. It supports typed holes and includes a clipboard for pasting copied items into type-compatible locations. The upper right corner displays the type of the code block at the cursor, with a keyboard shortcut guide below for quick reference.

Figure 5 illustrates the structured yet intuitive nature of expression entry and editing within ElmSr. Expression input appears to users as a natural text entry process, allowing them to type and modify code seamlessly. However, the underlying structure being manipulated remains a well-formed, typed AST at all times.

This ensures that every modification aligns with the constraints of Elm's type system. The figure presents a sequence of editing states, where the cursor navigates through subexpressions, enabling context-aware operations such as wrapping expressions in let-in, extracting values into function arguments, or defining new bindings. For example, a numeric constant within an expression can be elevated to a function argument using the **Shift + P** key combination.

4. Multilingual support in ShapeCreator

An ad-hoc framework was developed specifically to support translation of ShapeCreator. Strings were extracted and a template translation module with a list of translation pairs. Translators duplicate the template module, or an existing module, and translate the 100 strings. To accommodate variable word or phrase lengths, each translated string is paired with a font size, to allow the developers to adjust sizes in a consistent way for grouped interface elements. When users select a different language using

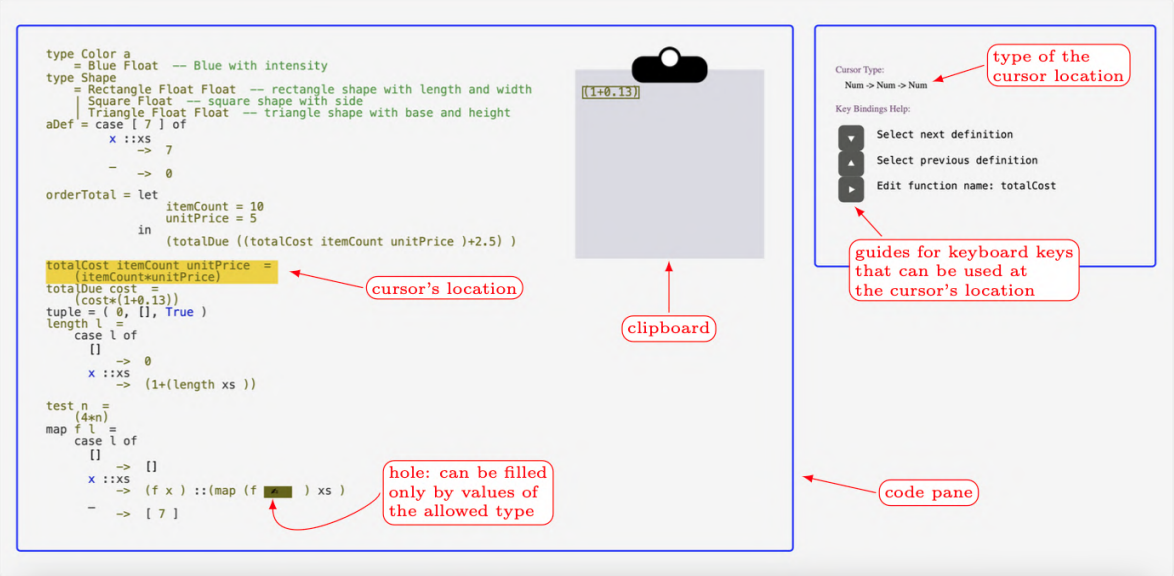

Figure 4: ElmSr's graphical user interface.

the menu of clouds, the appropriate list of pairs is converted into a dictionary which is passed in to a translation function used by all interface elements. Figure 6 shows the interface in the Korean translation. The meaning of the code does not change, as you can verify by comparing this figure with figure 2. Currently, this will be useful for students copying their code into the WebIDE in order to add animations or interactivity.

The latest development version of ShapeCreator is available at STaBL Foundation.

5. Future work

5.1. Curriculum development

We are in the process of releasing French versions of our six lessons on YouTube. We have plans to record several other languages, for which we have confident native speakers. Not everyone is comfortable making videos, however, and contrary to our initial expectations, our videos are mostly used by teachers, many of whom are equally comfortable watching videos in English. We will have to evaluate whether having non-English videos results in greater teacher recruitment.

The book *Creating with Code* [2] is published under a Creative Commons license, so any educator could translate (and adapt) sections relevant to their teaching, but it would be a big undertaking to translate the whole book. However, most students today prefer to learn from videos, and it is easy for native speakers to produce short videos. So that is probably a better use of resources than translating the whole textbook.

5.2. Structure editor

Introducing ElmSr will aid language support in two ways. Being a structure editor, it will be easy to support translation of the language, libraries and interface. However, support for creating translations can also be added. Currently, supporting new languages in ShapeCreator requires manual updates to translation files within the codebase, with translations provided as pull requests to the GitHub repository. This process is a barrier for educators and translators unfamiliar with version control systems like Git. To address this, ElmSr could include a dedicated translation facility, allowing users to create translations directly within the editor. Translators could define localized terms for exposed functions, data types, and strings in shared modules using the intuitive interface of the editor. This

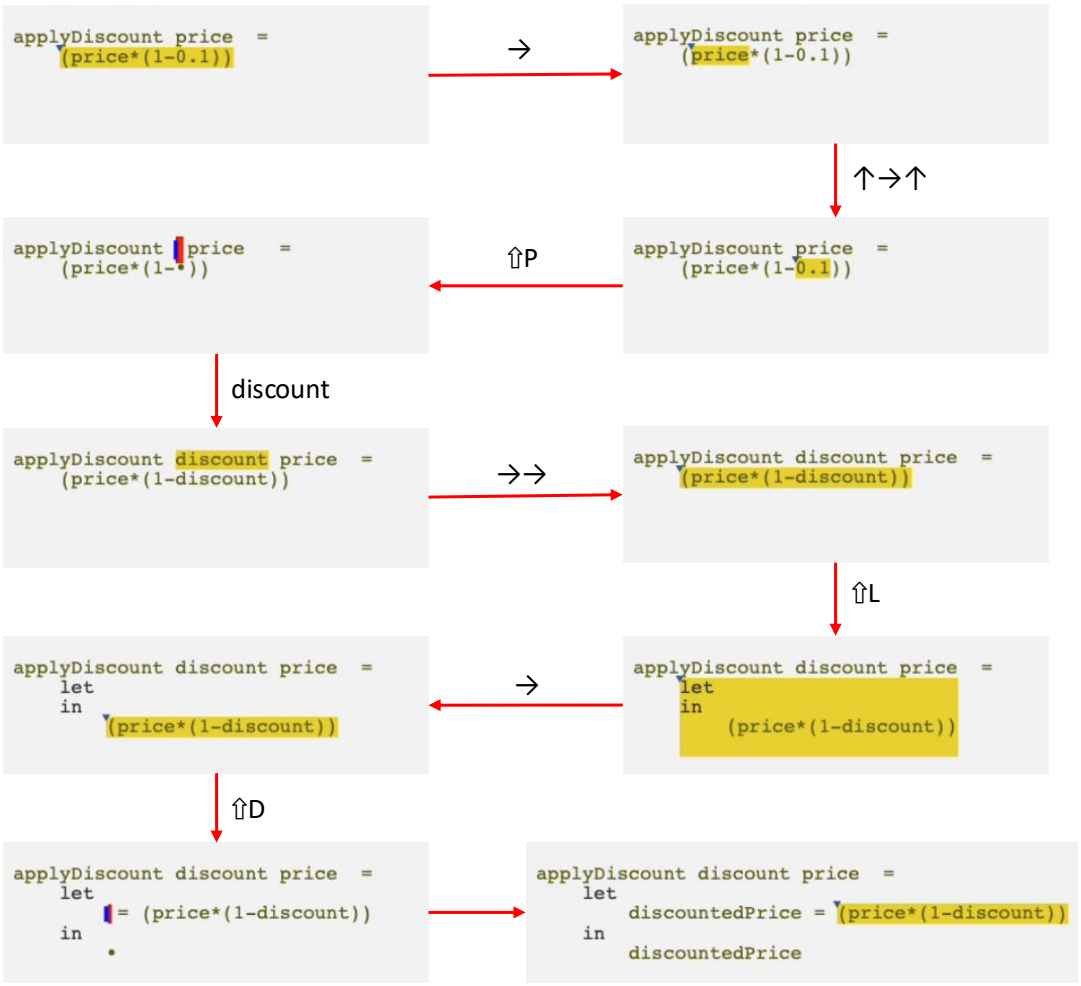

Figure 5: ElmSr edit actions. The arrows indicate transitions between states, with key combinations shown on the arrows representing the keys pressed to reach the next state.

approach would enable teachers and translators to easily add translations and expand the ecosystem's accessibility to a wider audience.

5.3. Evaluation

With our partners, we are planning tests for ShapeCreator with students using Mi'kmaq, Tibetan, and Tamil, in the context of immersion programs for Indigenous students, minority language speakers, and Tamil-speaking children with very limited knowledge of English. This informal validation will be used to refine the curriculum used in each situation. Formal evaluations will take place once the curriculum has been standardized, in one or two years.

6. Recommendations

Native support Although further research is required to strengthen the evidence that native-language support in a programming language leads to accelerated learning, based on the combination of the evidence we have, and the well-known lack of diversity in the software industry, we recommend that all programming languages widely used in education should be translated into non-English languages.

Code switching Based on ample evidence from multiple subjects that code switching is beneficial to students, we recommend that teachers be trained to teach children that people have developed

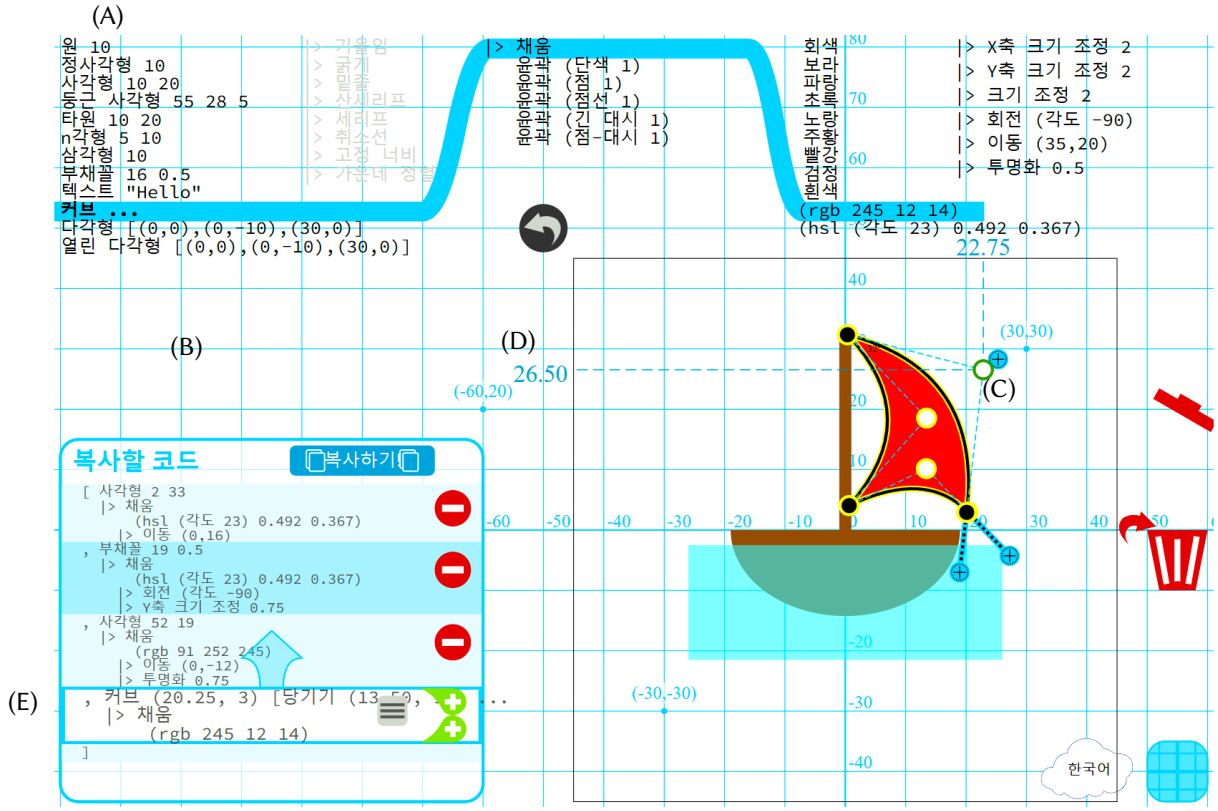

Figure 6: The ShapeCreator in Korean mode, showing (A) the list of stencil functions translated into Korean, (B) the graph paper is active, which can be set independently of the language, (C) showing the curve editing mode, with a pull point selected, (D) the y-coordinate for the pull point being edited (the x and y coordinates of the anchor or control points are always shown, to emphasize the importance of Cartesian coordinates in creating graphics, and (E) the resulting code which is preserved when switching languages.

many different types of language for communication, including human languages, mathematical languages and programming languages. Using a combination of the best languages to express ideas will lead to better communication and better understanding.

7. A call for collaboration

Our main contribution of this paper is a framework for translating a text-based language into non-English languages. Using this framework, curriculum developers can support introductory text-based coding in any language. Using the translations already available, teachers can support students in the language in which they are more comfortable. For students who *want* to learn another language, they will be intrigued to create code in one language and have translated into any of the other languages at the click of a button. This puts students in control of their learning environment, and allows them to make the transition at their own pace.

The impact of these tools will depend on the number of languages supported. We therefore invite other educators and researchers to contact us to learn how to contribute additional languages, and access other tools.

Author Contributions

Conceptualization – Christopher Kumar Anand; methodology – Christopher William Schankula; formulation of tasks analysis – Kji-Wikuom Studios and Christopher William Schankula; software – Kji-Wikuom Studios and Christopher William Schankula; writing – original draft – Kji-Wikuom Studios and Juthada (Jamie) Malakulang; analysis of results – Christopher William Schankula and Kji-Wikuom

Studios; visualization – Christopher Kumar Anand and Juthada (Jamie) Malakulang; reviewing and editing – Juthada (Jamie) Malakulang and Christopher William Schankula. All authors have read and agreed to the published version of the manuscript.

Funding

This study did not receive any funding.

Data Availability Statement

No new data were created or analysed during this study. Data sharing is not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Acknowledgments

We thank the translators who contributed translations to MultiLingual ShapeCreator - Aadya Khanna, Bradley Chamberlain, Cindy Zhu, Hannelore Anand, Melissa Ho, Omid Isfahani Alamdari, Sophia He, Tenzin Migmar, and Yara Alammouri. Also, teachers and students for their feedback on past versions, especially our high-school mentors and undergraduate volunteers.

Declaration on Generative AI

The authors have not employed any Generative AI tools in preparing this paper.

References

[1] S. Dasgupta, B. M. Hill, Learning to Code in Localized Programming Languages, in: Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale, ACM, Cambridge Massachusetts USA, 2017, pp. 33–39. doi:10.1145/3051457.3051464.

[2] C. Anand, G. Dulai, L. Yao, M. Arief, O. D'Mello, S. S. Menon, C. W. Schankula, Creating with Code: an Introduction to Functional Programming, User Interaction, and Design Thinking, version 1. ed., Fondation STaBL Foundation, [Louisbourg, Nova Scotia], 2023.

[3] D. Silver, M. Saunders, E. Zarate, What factors predict high school graduation in the Los Angeles Unified School District, volume 14, California Dropout Research Project Santa Barbara, CA, 2008.

[4] E. A. Silver, On My Mind: "Algebra for All"—Increasing Students' Access to Algebraic Ideas, Not Just Algebra Courses, Mathematics Teaching in the Middle School 2 (1997) 204–207. doi:10.5951/MTMS.2.4.0204.

[5] C. Kieran, Algebraic thinking in the early grades: What is it, The Mathematics Educator 8 (2004) 139–151.

[6] M. Driscoll, Fostering Algebraic Thinking: A Guide for Teachers, Grades 6-10, Heinemann, 361 Hanover Street, Portsmouth, NH 03801-3912 ($23), 1999.

[7] J. Solomon, Programming as a Second Language, Learning & Leading with Technology 32 (2005) 34–39. URL: https://files.eric.ed.gov/fulltext/EJ697292.pdf.

[8] S. Krishnamurthi, K. Fisler, Programming Paradigms and Beyond, in: S. A. Fincher, A. V. Robins (Eds.), The Cambridge Handbook of Computing Education Research, 1 ed., Cambridge University Press, 2019. doi:10.1017/9781108654555.014.

[9] E. Schanzer, K. Fisler, S. Krishnamurthi, Assessing Bootstrap: Algebra Students on Scaffolded and Unscaffolded Word Problems, in: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, ACM, Baltimore Maryland USA, 2018, pp. 8–13. doi:10.1145/3159450.3159498.

[10] G. Hong, J.-F. Yao, C. Michael, L. Phillips, A multilingual and comparative approach to teaching introductory computer programming, Journal of Computing Sciences in Colleges 33 (2018) 4–12.

[11] S. Alaofi, S. Russell, The Influence of Foreign Language Classroom Anxiety on Academic Performance in English-based CS1 Courses, in: Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research, ACM, Dublin Ireland, 2022, pp. 1–7. doi:10.1145/3555009.3555020.

[12] S. Alaofi, S. Russell, The Use of English Language to Teach CS1 to Non-Native English Speakers: Students Perspective, in: Proceedings of the ACM Conference on Global Computing Education Vol 1, ACM, Hyderabad India, 2023, pp. 15–21. doi:10.1145/3576882.3617931.

[13] P. G. Feijoo-Garcia, K. McNamara, J. Stuart, The Effects of Native Language on Block-Based Programming Introduction: A Work in Progress with Hispanic Population, in: 2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), IEEE, Portland, OR, USA, 2020, pp. 1–2. doi:10.1109/RESPECT49803.2020.9272513.

[14] A. G. Soosai Raj, K. Ketsuriyonk, J. M. Patel, R. Halverson, Does Native Language Play a Role in Learning a Programming Language?, in: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, ACM, Baltimore Maryland USA, 2018, pp. 417–422. doi:10.1145/3159450.3159531.

[15] E. R. Dodoo, T. Nelson-Fromm, M. Guzdial, Teaching Computing to K-12 Emergent Bilinguals: Identified challenges and Opportunities, ACM, Pittsburg, 2025. doi:https://doi.org/10.1145/3641554.3701889.

[16] T. M. Khan, S. W. Nabi, English versus Native Language for Higher Education in Computer Science: A Pilot Study, in: Proceedings of the 21st Koli Calling International Conference on Computing Education Research, ACM, Joensuu Finland, 2021, pp. 1–5. doi:10.1145/3488042.3488070.

[17] P. J. Guo, Non-native english speakers learning computer programming: Barriers, desires, and design opportunities, in: Proceedings of the 2018 CHI conference on human factors in computing systems, 2018, pp. 1–14.

[18] B. A. Becker, Parlez-vous Java? Bonjour La Monde!= Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers, in: PPIG, 2019.

[19] D. B. MacDonald, G. Hudson, The Genocide Question and Indian Residential Schools in Canada, Canadian Journal of Political Science 45 (2012) 427–449. doi:10.1017/S000842391200039X.

[20] A. Berciano, A. Cuida, M.-L. Novo, The importance of coding and translation between programming languages in sequential activities of pre-service teachers: an approach, Education and Information Technologies (2024). doi:10.1007/s10639-024-13092-1.

[21] M. U. Bers, Coding as another language: a pedagogical approach for teaching computer science in early childhood, Journal of Computers in Education 6 (2019) 499–528. doi:10.1007/s40692-019-00147-3.

[22] I. Ruby, B. Krsmanovic, Does learning a programming language require learning English? A comparative analysis between English and programming languages, in: EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE), 2017, pp. 420–427.

[23] V. Donzeau-Gouge, G. Huet, G. Kahn, B. Lang, J. Levy, A structure-oriented program editor: a first step towards computer assisted programming, IRIA. Laboratoire de Recherche en Informatique et Automatique, 1975.

[24] D. B. Garlan, P. L. Miller, GNOME: An introductory programming environment based on a family of structure editors, SIGPLAN Not. 19 (1984) 65–72. doi:10.1145/390011.808250.

[25] M. Voelter, T. Szabó, S. Lisson, B. Kolb, S. Erdweg, T. Berger, Efficient development of consistent projectional editors using grammar cells, in: Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Association for Computing Machinery, New York, NY, USA, 2016, pp. 28–40. doi:10.1145/2997364.2997365.

[26] P. Voinov, M. Rigger, Z. Su, Forest: Structural Code Editing with Multiple Cursors, in: Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Association for Computing Machinery, New York, NY, USA, 2022, pp. 137–152. doi:10.1145/3563835.3567663.

[27] C. Omar, I. Voysey, M. Hilton, J. Aldrich, M. A. Hammer, Hazelnut: a bidirectionally typed structure editor calculus, in: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, Association for Computing Machinery, New York, NY, USA, 2017, pp. 86–99. doi:10.1145/3009837.3009900.

[28] N. Osmani, ElmSr: A Structure Editor for Elm, Master's thesis, McMaster University, 2024. URL: https://macsphere.mcmaster.ca/handle/11375/29828.