Paper ID #37703

Can Natural Language Acquisition Theory Inform How Students Learn To Program?

Jose Alejandro Cabrera

Ashish Aggarwal

Ashish Aggarwal is an Instructional Assistant Professor of Computer Science in the Department of Engineering Education at the Herbert Wertheim College of Engineering, University of Florida. His research focuses on Computer Science Education and Learning Analytics where he studies the effectiveness of different learning approaches on students' learning outcomes and performance in programming courses.

Can Natural Language Acquisition Theory Inform How Students Learn To Program?

Abstract

Second Language Acquisition is a field of Linguistics that studies how humans learn additional languages after their early years. This field is heavily tied to cognitive science and has focused on both the neurological and pedagogical aspects of language acquisition. Over the years, studies have focused on acquisition models that have enhanced the way we teach and learn languages. Naturally, these findings can also be applied to other disciplines and be contextualized within their areas of knowledge.

Programming and natural languages share design features in various areas such as their foundations, syntax, and semantics. Comparing the nature of both types of languages is key to have a deeper understanding of the acquisition models that both fields have developed over time. Moreover, analyzing the way learning works in the context of both programming and natural languages can yield potential improvements on effective language learning through improved instruction and pedagogy.

This paper describes and explores the similarities and differences of programming and natural language acquisition based on their foundations, syntax, and semantics. These comparisons give a theoretical foundation for a further analysis of the similarities and differences in acquiring programming and natural languages. Several key points are highlighted, backed by acquisition theory and other studies; namely in the context of acquisition stages, learning components, factors and elements that benefit or hinder the acquisition process, etc. Likewise, several elements that differ between both models are emphasized, namely those that have different context when applied to programming or natural languages, respectively.

All these findings give rise to several implications that connect back to practices that may enhance the way knowledge is imparted on both fields, especially at early stages of acquisition. Various study-backed recommendations are also listed in order to provide more effective methods of teaching introductory courses in Computer Science, highlighting the inherent advantages of the field as well as covering some weaknesses that teaching the subject can have.

1 Introduction

Language learning has been a topic of interest for many researchers throughout the past decades. [1, 2, 3] Factors like technological advancements and the sudden shift towards globalization in the modern world have enabled the field of Linguistics to study the intricate process of language acquisition and its neurological, psychological, and pedagogical aspects. Language acquisition is still a developing branch of Linguistics, lacking strong conclusive results on several research questions, such as the extent of bilingual advantage [4], the nature of code-switching [5], etc. However, various models of language learning [6] have been developed, with similar factors and stages of development.

The increasing economic value of multilingualism in modern society has increased the demand for language learning across the globe. [7] Likewise, the rising accessibility of resources such as applications like Duolingo, Babbel or contact with native speakers through written and/or spoken communication platforms (WhatsApp, Skype, etc.) has given people more means to use, practice, and learn a foreign language; whether as a hobby or as a necessity.

Parallel to this, in the field of Computer Science, various programming languages have been developed throughout the decades as a tool for software development and problem-solving. It is common for a software developer to eventually acquire multiple programming languages (often related to functionality, e.g., frontend, backend, mobile app development, etc.). This necessity makes the effective acquisition of various programming languages a useful asset for those working in the field. Improvements in the way introductory courses teach the basics of programming and the approaches that instructors use to achieve this goal could be essential for future generations of learners [8].

Since programming languages behave similarly to their natural homologs, there is a need to analyze how humans acquire both types of languages. Making a cross-comparison between their foundations, elements, and acquisition could be a useful tool to understand how do we learn to program. This comparative analysis can be supported by the research that has been done for both types of languages in isolation. Ultimately, this has the potential to inform about additional characteristics of language learning, which can influence the acquisition processes of both programming and natural languages.

2 Theoretical Framework

It is crucial to establish the areas of comparison to structure a further discussion on the acquisition models. A linguistic approach to this task would be to divide each section according to the level of complexity of the structure. That is, going from the **foundations** of the language to their **phonology** (set of sounds), **morphology** (word building and classification), **syntax** (the grammatical interaction between words and word classes), **semantics** (the logical meaning embedded on sentences), and **pragmatics** (the way context contributes to meaning). It is worth noting that some branches will have little to no comparison due to the nature of programming languages: for example, since programming languages were never meant to be spoken, they lack any phonology. Likewise, morphology and pragmatics in programming languages work very differently from their natural language homologs, making their comparison an extended discussion, rather than a one-to-one analysis.

This structure leaves us with three major areas to analyze: the **foundations**, **syntax**, and **semantics** of the languages. The discussion of these branches will give us enough understanding of the nature of both types of languages to lay the groundwork for a major analysis of their acquisition, which is a topic linked to their application. Table 1 briefly lists significant similarities and differences in the areas of study, as mentioned above with respect to both programming and natural languages.

Sub-field	**Similarities**	**Differences**
Foundations	Primary purpose: communication and shaping of thinking. Family trees and language variation present in both types of languages	The difference in the recipient (human to human versus human to machine). Variation in programming languages showing additional non-linguistic functionality.
Syntax	Shared rules and syntactic analysis (word classes, subject/predicate, etc.)	Word classes are grouped for function in programming languages (instead of grammatical category)
Semantics	Compositional semantics work similarly (sentences as a sum of propositions)	Lexical semantics absent in programming languages Lack of ambiguity and double meaning in programming languages' semantic interpretation

Table 1: Similarities and differences of the foundations, syntax, and semantics of programming and natural languages.

2.1 Foundations

Fundamentally, both natural and programming languages exist for communication. Natural languages were developed as a tool for interaction between members of a community, conveying necessary information for complex coordination and survival techniques. Also, humans used languages as a tool for transmitting knowledge from older to newer generations without first-hand experiences through storytelling and physical documentation (i.e., written languages). Similarly, programming languages were developed as a bridge between human and machine interaction: something that would facilitate the integration of computing into everyday life. Without a way to make low-level machine code intelligible for computer scientists, it would be nearly impossible to build the information structures humans have today.

Likewise, both types of languages shape the way we think. Natural languages gave humanity a way to take abstract thoughts and describe them in a way that can be understood by others (given that both speaker and recipient speak the same language). Programming languages, on the other hand, were the bridge for the application of computational thinking (from an abstraction of problem-solving ideas to automation of the algorithms needed to perform a given task) [8].

This distinction of the intended recipient is what draws the most differences between the types of languages (Figure 1). Computers are limited to interpret the data as a set of instructions, whereas humans can convey and understand emotions, beliefs, and ideologies through language. Similarly, natural languages emerge as a necessity to interact with other humans: a study known as the "Conditioned Head Turn Procedure", demonstrates how a child's brain can recognize speech patterns from the earliest stages of their life [9]. Compared to learning a natural language, learning to code may not be an essential part of every human's life. It is essential to highlight this feature, as other differences that are present in both types of languages (which we will discuss in further sections) are derived from a change in the recipient.

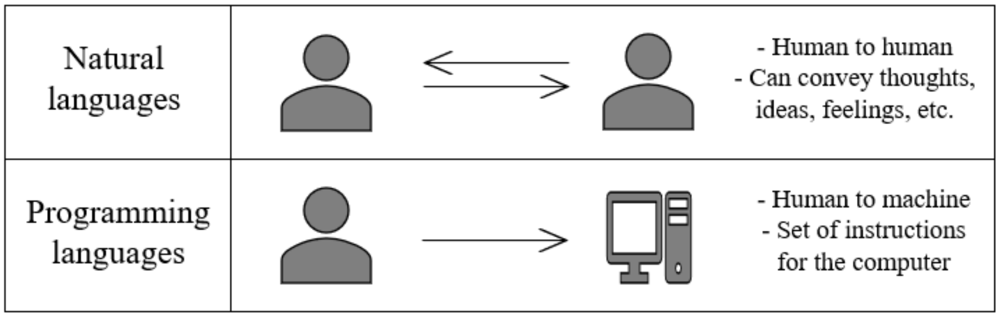
Natural languages
- Human to human
- Can convey thoughts, ideas, feelings, etc.
Programming languages
- Human to machine
- Set of instructions for the computer

Figure 1: Foundational difference of programming and natural language regarding the intended recipient

Another fundamental nuance between both types of languages is the relationship within language families. Language variation is a process in which small changes in a given language lead to what is referred to as "dialects" or "accents" [10]. As these differences widen due to historical and anthropological circumstances, new languages eventually emerge, being linked by a common ancestor (which, by the time this happens, is usually not spoken anymore). A way to group languages that derive from the same common ancestor is by using language families. For example, languages like Spanish, Italian, Portuguese, French, Romanian, et al. are part of the Romance family of languages, with Latin being their common ancestor [11]. Programming languages emulate dialects with their updates that regularly change some of their syntax and semantics, and have family trees, often more complex than their natural language homologs. This is because the process of deriving a new programming language can also involve the inheritance of non-linguistic features present on its common ancestors (i.e., coding functionality). It is worth noting that, since programming languages are developed mostly for a specific niche (e.g., to run natively on a given operating system, enhance the performance of an older language, etc.), it is common to see languages having more than one direct ancestor; contrary to natural languages in which overlaps of various languages from distinct families are rare (usually in the form of pidgins and creoles [10]).

2.2 Syntax

In Linguistics, syntax is the branch that studies the classification and relationship of words in a language. Basics of syntax include:

- **Word classes:** Also known as parts of speech, they are categories in which words with a similar grammatical function are grouped together, e.g., nouns, verbs, adjectives, adverbs, conjunctions, adpositions, etc [12].
- **Subjects and predicates:** The subject of a clause is the entity performing the action, while the predicate is whatever the subject does or is [12].
- **Heads and dependents:** The head of a clause is the word with the most significant semantic meaning of the sentence (e.g., in the sentence *"The dog was barking at the girl in front of him"*, the head of the clause is *"the dog"* since the clause revolves around it). The dependents are other pieces of information that accompany the head and provide complemental meaning to the sentence. Languages can either be head-initial (with the head at the beginning of a clause) or head-final (with the head at the end of a clause) [12].
- **Complements and adjuncts:** Dependents can either be complements, which are mandatory particles that complete the meaning of the head (e.g., for the sentence above, *"at the girl"* is a complement); or adjuncts, which are optional particles that specify the sentence within certain boundaries (e.g., for the sentence above, *"in front of him"* is an adjunct) [12].

Programming languages follow the same syntactic concepts of their natural homologs and can be effectively categorized and analyzed using these principles. This feature is what makes pseudocode as an adequate token replacement for a programming language, as it is relatively easy to translate the constituents that conform a line of code into a sentence. It is worth noting that word classes in programming languages may be related to functional aspects (what they do) rather than their grammatical purposes (where they appear). For example, in a programming language like C++, although pointers and constant attributes serve as adverbs (since they modify an adjective, which would be the data type the variable (noun) possesses), they are generally considered two separate concepts due to their different function and applications.

2.3 Semantics

In Linguistics, semantics refer to the meaning of a sentence, without considering its context or intent. This field of Linguistics is split between **lexical semantics** (the meaning of single words or small phrases) and **compositional semantics** (the meaning of phrasal expressions or sentences as a whole unit).

Lexical semantics in programming languages are almost impossible to study. This is because they operate based on two aspects: **sense** (the mental representation of the meaning in an expression) and **reference** (the concept this expression refers to). The nature of user-defined meaning within a programming language makes this analysis inconsistent. For example, function overloading in a programming language can disjoint the link between sense and reference without the use of documentation that explicitly defines their relationship.

As for compositional semantics, the analysis comes down to evaluating the truth value of a sentence as a sum of propositions. Propositions are expressions that carry a truth value, which is often linked to the syntactic relationships and constraints of its constituents. For example, the sentence *"The young man dances"*, there are two propositions: *"The man is young"* and *"The man dances."* The sentence *"The table dances"* is not a proposition, since it does not have a truth value (as an inanimate noun cannot perform a verb associated with animate nouns). It is worth noting that ambiguous sentences (conditionals, questions, imperative commands, etc.) can be rephrased to determine if they abide by the compositional semantic rules. Programming languages follow compositional semantics just like their natural homologs, with errors in syntax making expressions fail the proposition evaluation.

The major difference in this field is the interpretation of the recipient when dealing with semantic errors. Machines are not able to interpret small mistakes that would deem the expression as infelicitous (semantically inappropriate), with some noteworthy exceptions like HTML, in which the absence of a closing tag may not affect functionality. Humans can overcome the offset of a relatively small mistake and try to deconstruct the intended meaning of the expression, whereas computers will often mark the error as unacceptable.

Another nuance is the presence of multiple meanings in each expression, something that programming languages lack. Natural languages can use techniques such as slang, jargon, variation in intonation, argot, etc. to inflict different meanings into the same expression depending on the common understanding between the speaker and the recipient. Machines do not possess this discerning ability and are limited to understand any phrase or a sentence in only one way.

This comparison between the core components of natural and programming languages paves the way for an analysis of their acquisition models and assumptions. The similarities between both types of languages will be emphasized, as this will be key for the conclusions and implications of the paper.

3 Comparing language acquisition

Language acquisition can be defined as the process in which an individual develops proficiency (i.e., mastery in an area of communication, such as speaking, listening, etc.) in a formerly unknown language. Although learning to code entails knowledge that falls beyond the scope of this definition, such as design features of computing (e.g., the use of binary, code architecture, etc.). Table 2 mentions critical aspects of the similarities and differences of language acquisition.

Similarities	**Differences**
The principles and milestones of acquisition are similar in both types of languages.	Natural languages have a more significant emphasis on social context and interaction when learning.
They both have two independent learning components (passive and active) that must be acquired to achieve proficiency.	Programming languages have features that make immediate feedback inherently. easier to receive than natural languages.
Both are influenced by similar factors (motivation, attitude, cognitive style, etc.) that can enhance or impede the acquisition process.	The processes of acquisition and the resulting "proficiency" differs from both types of languages.

Table 2: Similarities and differences in the acquisition of programming and natural languages.

3.1 Similarities

- **Principles and milestones:** The first similarity in acquisition comes from the foundations of both types of languages. Since they both serve the same purpose (communication), they also share similar milestones when measuring the retention of the reader. Barmpoutis defines these milestones for programming languages as three stages of learning (acquiring, consolidating, tuning) with two types of knowledge: **declarative** (stating the concept to be learned) and **procedural** (internalizing the concept until it becomes "muscle memory") [13]. Natural language acquisition also has a similar model with five stages [14]. Table 3 further discusses the stages of each acquisition development model.

  Both models are very similar, with an emphasis on learning the basic concepts of the language, then building structures that become more complex as the learner gets accustomed to the nuances of the language, and finally achieving a peak in which proficiency is achieved by polishing the concepts that are not already mastered. The similarity in the milestones that define acquisition is essential, as this can be a useful metric to measure the proficiency of the learner.

Programming language learning stages	Natural language learning stages
Acquiring: the learner is focusing on declarative knowledge to understand the basic concepts of the language	**Silent:** the learner spends time learning basic vocabulary and getting accustomed to the language as a whole.
Consolidating: a leap between declarative and procedural knowledge is done, so the learner establishes their knowledge	**Early production:** the learner has some vocabulary, and it starts to form basic grammatical structures.
Tuning: the learner polishes any mistakes when retrieving the concepts until it becomes second nature.	**Speech emergence:** the learner learns complex structures and consolidates most of the critical vocabulary needed.
	Intermediate fluency: the learner consolidates the complex structures, thinking in the target language becomes common.
	Advanced fluency: the learner tunes any concepts needed for mastery.

Table 3: Language acquisition development models.

- **Learning Components:** Our second similarity is the presence of two independent components of learning the language: **passive** and **active**. Passive skills refer to the ones in which the brain processes information (i.e., reading, listening), and active skills refer to the active neurological production of information (i.e., speaking, writing). It is worth noting that although programming languages lack a spoken variant of these components, this does not impede the comparison (due to many natural languages lacking a writing variant). These components are directly tied to the stages of learning, as a learner must be accustomed to understanding a language before being able to reproduce any of its features.

- **Factors that affect language learning:** The last similarity to be discussed in this paper is the set of factors that can affect the language learning process (either positively or negatively), defined by Khasinah [15]. Table 4 contains the factors that are present in both natural and programming languages.

Factor	**Effect**
Motivation	A learner who has a reason (either by necessity or for curiosity) to learn a language is prone to learn faster.
Attitude	Positive or negative views on the target language and the culture surrounding it make language learning easier or more challenging, respectively.
Aptitude	Certain inherent skills (namely memorization and pattern recognition) can give an edge for learning a new language.
Self-esteem	Learners who think negatively of their ability to acquire a language will have more difficulties when learning.
Cognitive style	Regardless of the cognitive style of the learner (visual, auditive, kinesthetic), picking the style that aligns with them dramatically enhances language learning.

Table 4: Similar factors that affect language acquisition.

The acquisition of programming languages, similar to their natural homologs, shares these factors. With some small exceptions and nuances (mainly coming from the fact that programming language learning also entails other technical components of the language, namely functionality, and purpose in a system), the recognition of these factors can dramatically enhance the acquisition process of a learner.

3.2 Differences

- **Social context:** A significant difference between natural and programming languages is how the former has an embedded element of a social context that must be present to acquire the principles of the language properly. In the same study from Khasinah [15], in which they list several factors that influence natural language acquisition, there is an emphasis on social context. These factors are listed on Table 5.

  As discussed in section 2.1, the change in the intended recipient of the language is responsible for a difference in the type of information that can be conveyed through language. For natural languages, human-to-human interaction also has societal and cultural components that explain these factors. As for programming language acquisition, the human-to-machine exchange does not allow for social context to interfere with learning and to master the syntax and semantics of the target language.

Factor	**Effect**
Extroversion	Extroverted and introverted learners tend to develop their active and passive skills with more ease, respectively.
Anxiety	An anxious learner may feel uncomfortable using their active components of language learning, thus hindering their ability to learn these skills.
Inhibition	A learner that inhibits and heavily punishes their mistakes when producing speech, similarly to anxiety, will effectively hinder their ability to acquire language.

Table 5: Social factors that only affect natural language acquisition.

- **Feedback:** When phasing from the declarative to the procedural knowledge stages in the language learning model, programming languages use the immediate feedback received from the applications of their code to consolidate the concepts to learn. The feedback sources for natural languages, on the other hand, are somewhat limited in comparison. Not every learner can count on resources such as a native speaker to practice with or substantial documentation, especially when learning endangered languages. In contrast, programming languages can always test and debug their code to learn and polish their skills. It is worth noting that programming concepts such as a good code architecture, or optimization of resources such as time, computing power, or memory allocation fall beyond the scope of this nuance.

- **Thinking and "proficiency":** Although the milestones and principles of both languages are similar, the definition of "proficiency" differs between programming and natural languages. Programming language proficiency is tailored towards problem solving, optimization, and the use of computational thinking to develop the necessary algorithms and systems that will effectively give the desired output to the programmer. Natural language proficiency, on the other hand, is mainly tied to the articulated fluency of its passive and active skills (reading, writing, listening, and speaking). Having a fast reaction to these stimuli is more important than finding an optimized way of communicating. Additionally, features like register, intonation, and other forms of speech must be considered and adjusted throughout any conversation.

4 Implications

Comparing and analyzing the nature and acquisition of programming and natural languages is crucial for having a deeper understanding of the elements that can be integrated into developing proficiency and enhancing the ways in which they are taught. Likewise, differences in both models can help with the optimization of the areas in which the type of language differs from one another. For example, some natural languages have both spoken and written forms, adding a layer of complexity (since now the learner also has to consider the sounds of the language and their features e.g., intonation, pronunciation, etc.) that must be addressed in a way different than programming languages, which are only written. Some key implications of the comparison are:

- Since programming languages can be linguistically analyzed (for example, within the areas of syntax and semantics, as shown in 2.2. and 2.3.), it would be useful to implement basic concepts of these features in introductory-level courses. This would give the learners an ability to further understand how a specific programming language behaves. For example, when thinking about the syntax of a programming language, it would be useful to recall some linguistic concepts such as the head (the core of a sentence) and the dependents (auxiliary information for the head) of a sentence.

- Knowing that the stages of language acquisition development are similar in both types of languages [13, 14], it could be helpful to stress the importance of **declarative** and **procedural** knowledge since it is a relevant aspect of both models. For example, in the phenomenographic study done by Eckerdal and Berglund [16] performed at the University of Uppsala in Sweden, there was a recurrent struggle amongst students tied to a lack of procedural knowledge (referred in the paper as *canonical procedures*). For instance, if a learner of C++ cannot recall how to construct and use pointers with ease, it will hinder their ability to perform certain tasks that involve heavy use of pointers (e.g., passing objects to a function by reference).

- The shared factors for language acquisition can be beneficial for programming language courses. Profiling the students' motivation, attitude, aptitude, and cognitive style at the beginning of a course can help in how it is taught by giving helpful resources that accommodate the needs of each student. On a similar note, natural languages should also emphasize on the social context of learning a language, such as using cultural references and casual conversations to immerse the learner into the target language (and the culture surrounding said language). This way, a learner can take advantage of as many factors that enhance language acquisition as possible.

- The inherent feedback features of programming languages (e.g., compiler errors, integrated IDE spellcheckers, etc.) can be used as a powerful tool when practicing the target language. Making learners aware of these resources could make the learning process smoother, as these features are meant to help correct small mistakes, and learners may adapt quickly to the proper form of coding. A great example of this is how, in many programming languages, a semicolon is needed to end the command line. New programmers are usually reminded of the importance of the semicolon when compiler errors occur due to this issue and, with enough practice and exposure, they will instinctively put semicolons. Natural languages, on the other hand, may adopt similar strategies for feedback collection and data gathering that can help its learners on their consolidation and tuning of knowledge. For example, writing on an electronic device with spellchecking features can correct the spelling of the learner.

- Just like natural languages do, programming languages must develop both active and passive components of language acquisition simultaneously. For example, in an empirical study performed at the Pearl River Community College in Mississippi, the USA by A. G. Applin [17], there was a trend of increased performance in CS1 classes when students had the opportunity to modify templates on their programming assignments, compared to the convention of writing everything from scratch. A reason for this improvement may come from the dual task of both passive and active components of language learning, and more research needs to be done in order to confirm the effectiveness of this teaching method. One way to implement this could be providing snippets of code that follow “good programming practice” guidelines (e.g., clean code, a thorough program architecture, etc.), and making students pick up these habits by being regularly exposed to this design features, enhancing their ability to read through code and documentation.

5 Conclusion

The link between programming and natural languages and the applications of the knowledge between their similarities and differences is undeniable. The comparison between both types of languages in their foundations, syntax, semantics, and acquisition models yielded various implications that could improve the way both fields interact with learners. The implications of this comparison lie in several guidelines that could be implemented in both programming and natural language courses, with various examples of research studies that support the analysis. Improving the way introductory courses are taught will ultimately help cultivate a better understanding of computational concepts and constructs.

References

[1] R. C. Gardner and W. E. Lambert, “Attitudes and motivation in second-language learning.” 1972.

[2] B. McLaughlin, “Theories of second-language learning,” 1987.

[3] J. Arnold and M. C. Fonseca, “Multiple intelligence theory and foreign language learning: A brain-based perspective,” *International journal of English studies*, vol. 4, no. 1, pp. 119–136, 2004.

[4] M. Van den Noort, E. Struys, P. Bosch, L. Jaswetz, B. Perriard, S. Yeo, P. Barisch, K. Vermeire, S.-H. Lee, and S. Lim, “Does the bilingual advantage in cognitive control exist and if so, what are its modulating factors? a systematic review,” *Behavioral Sciences*, vol. 9, no. 3, p. 27, 2019.

[5] P. Auer, *Code-switching in conversation: Language, interaction and identity*. Routledge, 2013.

[6] S. Pinker, “Formal models of language learning,” *Cognition*, vol. 7, no. 3, pp. 217–283, 1979.

[7] M. Gazzola, “Gabrielle hogan-brun: Linguanomics: What is the market potential of multilingualism?” 2018.

[8] J. M. Wing, “Computational thinking and thinking about computing,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3717–3725, 2008.

[9] J. F. Werker, L. Polka, and J. E. Pegg, “The conditioned head turn procedure as a method for testing infant speech perception,” *Infant and Child Development*, vol. 6, no. 3-4, pp. 171–178, 1997.

[10] H. Dawson, M. Phelan *et al.*, *Language files: Materials for an introduction to language and linguistics*. The Ohio State University Press, 2016.

[11] D. M. Eberhard, G. F. Simons, and C. D. Fenning, “Languages of the world.” [Online]. Available: https://www.ethnologue.com/

[12] M. Tallerman, *Understanding syntax*. Routledge, 2014.

[13] A. Barmpoutis, “Learning programming languages as shortcuts to natural language token replacements,” in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, 2018, pp. 1–10.

[14] T. S. Team, “5 stages of second language acquisition: Resilient educator,” Dec 2018. [Online]. Available: https://resilienteducator.com/classroom-resources/five-stages-of-second-language-acquisition/

[15] S. Khasinah, “Factors influencing second language acquisition,” *Englisia: Journal of Language, Education, and Humanities*, vol. 1, no. 2, 2014.

[16] A. Eckerdal, M. Thuné, and A. Berglund, “What does it take to learn’programming thinking’?” in *Proceedings of the first international workshop on Computing education research*, 2005, pp. 135–142.

[17] A. G. Applin, “Second language acquisition and cs1,” *ACM SIGCSE Bulletin*, vol. 33, no. 1, pp. 174–178, 2001.