THE ROLE OF ENGLISH LANGUAGE PROFICIENCY AND LINGUISTICS IN PROBLEM SOLVING IN PROGRAMMING: A SYSTEMATIC LITERATURE REVIEW

by

TENARIO POWELL

B.A.Sc., Georgia Military College, 2019

MSIT, Kennesaw State University 2021

MSCYBR, Kennesaw State University, 2023

MBA, Southern New Hampshire University 2026

A Research Paper Submitted to the School of Computing Faculty of

Middle Georgia State University in

Partial Fulfillment for the Requirements for the Degree

DOCTOR OF SCIENCE IN INFORMATION TECHNOLOGY

MACON, GEORGIA,

2026

The role of English language proficiency and linguistics in problem-solving in programming: a systematic literature review

Tenario Powell, *Middle Georgia State University*, tenario.powell@mga.edu

Abstract

This systematic literature review explored how English language proficiency and linguistic factors influence problem solving in programming contexts. Programming languages, documentation, and development tools are primarily presented in English. Drawing on cognitive load theory and linguistic perspectives; the review analyzes how vocabulary knowledge, syntax interpretation, and semantic understanding affect a learner's ability to comprehend programming tasks, interpret compiler messages, and apply logical reasoning. The literature indicated lower levels of English proficiency may increase extraneous cognitive load, reduce comprehension accuracy, and make debugging and code construction more difficult for students in computer science education. The PRISMA method and thematic synthesis precisely address each research question. The evidence suggested language complexity within programming instruction, technical documentation, and collaborative communication environments can influence learning performance and participation. Findings highlighted the importance of linguistically supportive instructional strategies, clearer technical language, and scaffolding techniques to help reduce language related barriers. The review emphasized and addressed linguistic challenges can contribute to improved learning outcomes, more inclusive programming education environments, and stronger development of computational thinking skills among diverse learners.

Keywords: linguistics, programming, cognitive ability, and linguistic proficiency

Introduction

Programming is a highly analytical and linguistic process requires logical reasoning, abstraction, and attention to detail and all of which rely heavily on English-based syntax (Binkley et al., 2013). As programming education expands globally, students with English speaking backgrounds benefit from interpreting technical documentation, debugging code, and engaging in programming discourse communities. The increasing global demand for programming skills underscores the importance of understanding how English language proficiency influences learning outcomes, collaboration, and cognitive processing in software development contexts (Gordon, 2024). This study utilized a systematic research methodology to explore real world instances where English proficiency impacts programming problem solving within educational and professional settings. However, limited qualitative research exists exploring the experiences of programmers navigating language-based challenges (Alrasheed, 2021). Given this fragmentation, a systematic literature review (SLR) is needed to integrate diverse findings, evaluate methodological patterns, and identify unresolved gaps (Gordon, 2024). Unlike descriptive reviews, an SLR provides transparency, replicability, and structured synthesis across decades of research. Therefore, this review analyzed the current body of literature to determine how English proficiency influences programming comprehension, cognitive load, debugging, and collaboration. The systematic approach allowed for a comprehensive and consolidated understanding of how linguistic factors shape programming problem solving in educational and professional contexts (Gordon, 2024).

Although programming has become a universal skill, most programming languages and resources are designed for English speakers. English speaking programmers encounter obstacles which extend beyond syntax comprehension. They must translate, interpret, and contextualize meaning within a semantic linguistic framework. Previous research has identified correlations between English proficiency and programming performance, but limited qualitative exploration exists on how these barriers manifest in authentic learning and development environments (Gordon, 2024). Consequently, there is a gap in understanding the lived experiences of developers navigating both linguistic and technical challenges. This systematic study pursued to address this gap by investigating how English language proficiency influences cognitive load, problem-solving strategies, and collaborative practices in programming tasks. Well-established linguistics concepts articulate how humans often recognize subtle meanings in natural language prompts and instructions which can affect their performance on tasks in studies (Gordon, 2024). While quantitative research increasingly confirms the role of English proficiency in programming task performance; this is particularly in cognitive load and code perception qualitative understanding remains limited (Moreno, 2003).

Research Questions

R1 - What are the emerging themes found in the systematic literature on English language proficiency in computer science programs?

Since this study is qualitative and exploratory in nature, no formal hypotheses are proposed. Instead, it focuses on identifying recurring patterns, conceptual relationships, and emerging frameworks from observed literature.

Literature Review

Early research suggested programming is not solely a logical or technical activity; rather it involves substantial linguistic processing (Alrasheed, 2021). Learners must navigate English-structured syntax, keywords, documentation, and compiler messages, all of which require reading comprehension and familiarity with English grammar (Binkley et al., 2013). Most dominant programming languages embed English vocabulary and morphological patterns. Code interpretation involves the same linguistic mechanisms used in reading structured text (Gordon, 2024). Preliminary findings across computing education, psycholinguistics, and human computer interaction indicate English proficiency influences comprehension, debugging, collaboration, and access to learning materials. This preliminary review categorized key themes language barriers, cognitive load, code comprehension, and the linguistic features of programming to establish the need for a full systematic review.

Initial evidence across computing education research suggested English proficiency plays a significant role in programming performance. Multilingual students often encounter challenges in interpreting documentation, technical vocabulary, and error messages are typically written exclusively in English (Alrasheed, 2021). According to study, error messages depend extensively on specialized vocabulary and syntactic indicators; which poses challenges for students who are required to interpret unfamiliar terms (Gordon, 2024). These linguistic demands can slow comprehension and increase the cognitive effort needed to complete programming tasks. Grammar and vocabulary proficiency are predictive of students' ability to identify logical relationships, understand code, and articulate problem-solving steps (Alrasheed, 2021). Since programming languages draw heavily from English grammar, learners with lower English proficiency may misinterpret common keywords, conditionals, or natural-language comments. Preliminary findings therefore position language proficiency as a mediating variable which shapes abstraction, reasoning, and comprehension. Even when students possess strong logical skills, reduced English proficiency can hinder their programming performance (Gordon, 2024)

Research proposed that the use of complex language in conjunction with technical content may heighten the risk of cognitive overload (Moreno, 2003). Early findings further evoked language-heavy explanations and verbose documentation heighten extraneous load and slow conceptual transfer (Durán et al., 2022). From a preliminary perspective, these results support the hypothesis by reducing linguistic barriers may free cognitive resources, aiding comprehension and improving debugging efficiency.

Preliminary Coding Domains for the SLR

The LMCL-P framework will guide the coding and analysis phases of the future systematic review. Three preliminary domains have been identified:

1. **Linguistic Mediation** – including vocabulary comprehension, mental translation, and documentation navigation.
2. **Cognitive Load Effects** – including extraneous load from complex language and its influence on debugging and code construction.
3. **Problem Solving Behaviors** – including comprehension patterns, reasoning pathways, communication practices, and error-correction strategies.

These domains allow the SLR to compare diverse methodologies while synthesizing convergent themes across studies.

The LMCL-P framework will guide the coding and analysis phases of the upcoming systematic review by providing a structured method for examining how language, cognition, and problem solving interact in programming contexts. This framework ensures patterns across studies can be analyzed consistently, allowing linguistic and cognitive mechanisms to be interpreted through a common analytical structure. Research in computing education exemplifies learners are strongly influenced by language complexity, comprehension demands, and the ability to interpret technical terminology. Which reinforces the importance of a framework like LMCL-P for organizing and analyzing evidence.

For example, novice programmers experience higher cognitive load and reduced accuracy when confronted with complex or poorly structured language in programming tasks (Becker et al. 2019). Three analytic domains have been identified within the LMCL-P structure. The first domain, linguistic mediation, examines how vocabulary comprehension, mental translation, and navigation of English based documentation influence learners as they interpret programming tasks. The second domain, cognitive load effects, focuses on how linguistic complexity produces extraneous cognitive effort affects debugging, code reading, and code construction. The third domain, problem solving behaviors, includes comprehension patterns, reasoning strategies, communication practices, and error correction behaviors that emerge during programming tasks. Together, these domains ensure that the LMCL-P framework captures the linguistic and cognitive factors most relevant to understanding how learners process and solve programming problems.

Research Methodology

This study employed a systematic literature review (SLR) methodology to examine how English language proficiency influences programming comprehension and problem-solving. Systematic reviews are widely used in computing education and information technology research because they allow researchers to collect, evaluate, and synthesize evidence in a transparent manner (Kitchenham & Charters, 2007). Unlike traditional narrative reviews, an SLR follows a structured protocol which documents the search process, screening procedures, and study selection criteria. The methodological framework used in this study follows the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA 2020) guidelines. Which provide a standard for identifying, screening, and selecting scholarly studies (Page et al., 2021). The PRISMA methodology improves transparency by requiring researchers to document each stage of the literature search and selection process

The systematic review process consisted of six major stages:

1. Formulated research questions
2. Identified relevant academic databases
3. Developed and applying search strings
4. Screened studies through title and abstract review
5. Conducted full-text eligibility assessment
6. Extracted data and synthesizing findings thematically

This structured process ensures the systematic literature review captures relevant research across computing education, linguistics, and cognitive science while minimizing selection bias.

Barrier Category	Description	Examples in Programming Context
Cognitive Barriers	Barriers related to working memory limits, cognitive load, and processing complexity.	• Working-memory overload • High intrinsic load • Debugging complexity • Algorithmic reasoning strain
Linguistic Barriers	Barriers connected to language proficiency, terminology comprehension, and meaning construction.	• Low English proficiency • Mental translation of keywords • Difficulty with documentation • Misinterpreting error messages
Social Barriers	Barriers arising from social environment, peer interactions, and motivational climate.	• Classroom anxiety • Peer comparison • Lack of support networks • Stereotypes about belonging

Note. PRISMA 2020 flow summary of the identification, screening, eligibility, and inclusion process. *Adapted from Page et al. (2021).*

Databases

Searches were conducted across major academic databases, including:

- ACM Digital Library
- IEEE Xplore
- ScienceDirect
- SpringerLink
- Google Scholar
- ERIC

These sources were selected due to their coverage of computing education, computer programming, linguistics, cognition, and human computer interaction research.

Boolean search operators were used to combine keywords related to programming, language proficiency, and cognition. Search strings were adjusted slightly for different databases, but the following core search queries were used:

"English language proficiency" AND programming
"Programming comprehension" AND English
"Programming problem solving" AND language
"Software debugging" AND "language barriers"
"Cognitive load" AND programming AND language
"Multilingual learners" AND "computer science education"
"Code comprehension" AND "second language"
"Linguistics" AND "computer programming”

Search filters were applied to ensure relevance:

- Publication years: 2000–2025
- Peer-reviewed journal articles or conference papers
- English-language publications
- Studies involving programming, computer science education, or software development contexts

The search process produced 1,247 initial records across the selected databases.

Note. PRISMA 2020 flow summary of the identification, screening, eligibility, and inclusion process. *Adapted from Page et al. (2021).*

Studies were included in the systematic review if they met the following criteria:

1. Published in peer reviewed journals or conference proceedings
2. Focused on programming tasks such as debugging, code comprehension, or problem solving
3. Examined linguistic variables such as English proficiency, vocabulary knowledge, or language comprehension
4. Included participants such as students, novice programmers, or professional developers
5. Provided evidence using qualitative research designs
6. Published between 2000 and 2025
7. Available in full-text English format

Studies were excluded if they met any of the following conditions:

1. Non-empirical sources such as editorials, blogs, or opinion papers
2. Studies unrelated to programming or computer science education
3. Pure linguistics or ESL (English as a second language) research without programming context
4. Studies focused only on general computational thinking without language considerations
5. Publications lacking sufficient methodological detail

Applying these criteria ensured and complied with the final dataset included research directly examining the relationship between language and programming cognition.

PRISMA Explanation

The study selection process adhered to the four PRISMA stages: identification, screening, eligibility, and inclusion. During the identification phase, relevant records were gathered from selected databases. In the screening stage, studies were reviewed by title and abstract to remove those not addressing programming tasks, language proficiency, or computing education. The eligibility phase involved a detailed evaluation of full text articles based on inclusion and exclusion criteria. Finally, studies meeting all requirements were included in the final dataset. This systematic approach ensured only research directly examining the relationship between language and programming cognition was considered.

The initial database search produced 1,247 records across the selected databases. After removing 312 duplicate entries, a total of 935 unique studies remained.

During the title and abstract screening phase, studies which did not address programming tasks, language proficiency, or computing education contexts were removed. This stage resulted in 742 exclusions, leaving 193 studies for full-text review.

Full-text articles were evaluated based on the inclusion and exclusion criteria. During this stage, 178 studies were excluded due to:

- Lack of data
- No measurement of linguistic variables

- Absence of programming tasks
- Focus on general education rather than programming

After applying all screening criteria, 15 studies met the requirements and were included in the final systematic literature review.

PRISMA Flow Diagram

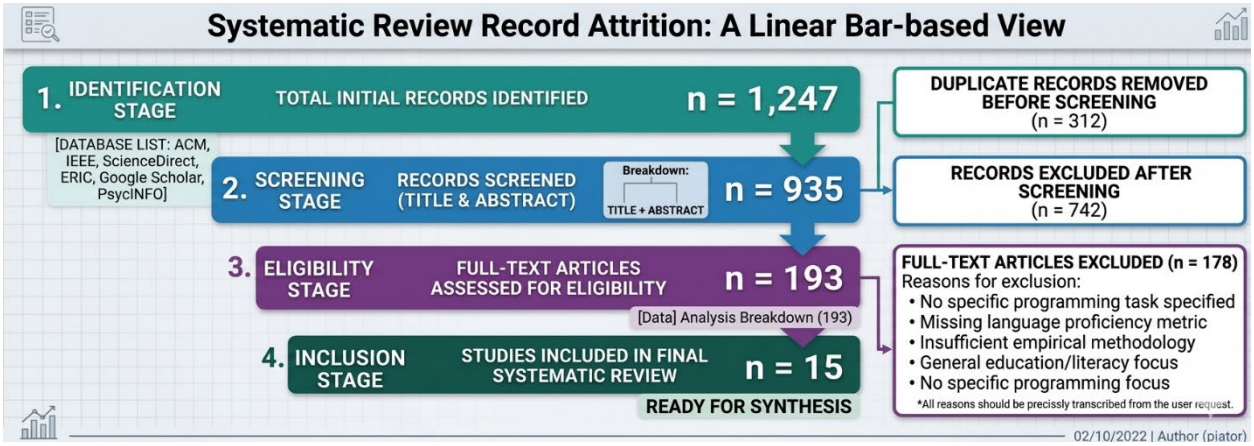

Figure 2
(Adapted from Page et al., 2021)

PRISMA 2020 flow diagram illustrating the study identification, screening, eligibility, and inclusion process used in this systematic literature review. The search across six academic databases produced 1,247 records, which were reduced to 935 after duplicate removal. Following title and abstract screening, 193 full-text articles were assessed for eligibility, resulting in 15 studies included in the final synthesis (Page et al., 2021).

A standardized extraction protocol captured:

- Article details (author, year, DOI, country)
- Participant characteristics (native vs. non-native English speakers, CS level)
- Programming task type (debugging, code reading, problem solving)
- Linguistic variable measured (English proficiency, vocabulary, grammar, error message readability, identifier clarity)
- Cognitive/measurable outcomes (accuracy, time on task, cognitive load rating)
- Instructional or design implications
- Study limitations

Study	Country / Context	Participants	Programming Task Type	Linguistic Variable Examined	Key Outcomes Measured	Key Findings	Theme
Lei et al., 2022	International CS education programs	English language learner (ELL) CS students	Programming coursework tasks	ESL (English as a second language) barriers and vocabulary comprehension	Learning outcomes, comprehension	ELL students experience vocabulary and syntax barriers reduce programming performance.	Challenges faced by ESL (English as a second language) learners in computer science

Study	Country / Context	Participants	Programming Task Type	Linguistic Variable Examined	Key Outcomes Measured	Key Findings	Theme
Alrasheed, 2021	University software engineering programs	Software engineering students	Problem-solving assignments	English proficiency (grammar, vocabulary)	task accuracy	English proficiency significantly predicts programming performance and academic outcomes.	Language comprehension in programming education
Alaofi & Russell, 2022	Introductory CS courses	CS1 students (ELL learners)	Exams and programming assignments	Language anxiety	Academic performance, anxiety levels	English-medium instruction increases language anxiety, reducing programming accuracy and confidence.	Challenges faced by ESL (English as a second language) learners in computer science
Becker et al., 2019	Programming education research	Novice programmers	Debugging tasks	Error message comprehension	Error correction rates	Complex compiler error messages significantly reduce debugging success rates.	Error message readability barriers
Lorås et al., 2021	Computer science education	Undergraduate CS students	Debugging exercises	Error message readability	Cognitive load, correctness	Simplified error messages improve comprehension and reduce debugging time.	Error message readability barriers
Durán et al., 2022	Computing education research	Novice programmers	Code reading tasks	Linguistic complexity and cognitive load	Cognitive load ratings, comprehension time	English-heavy programming documentation increases extraneous cognitive load.	Cognitive load and syntax interpretation
Hermans., 2020	Programming education for youth	Beginner programming students	Learning tasks in simplified programming language	Syntax complexity	Task completion rates	Simplified syntax improves comprehension for beginner learners.	Language comprehension in programming education
Avidan & Feitelson, 2017	University programming courses	Undergraduate CS students	Code comprehension tasks	Identifier naming clarity	Speed and accuracy of code understanding	Clear identifier naming significantly improves code comprehension.	Language comprehension in programming education
Binkley et al., 2013	Software engineering research	Student programmers	Code reading tasks	Identifier style and naming conventions	Comprehension effort	Linguistically complex identifiers increase cognitive effort during code comprehension.	Language comprehension in programming education
Ivanova et al., 2020	Cognitive neuroscience research	Adult programmers	Code comprehension experiments	Language-related neural processing	Brain activation patterns	Code comprehension activates brain regions associated with language processing.	Programming as a language-mediated cognitive activity
Becker et al., 2016	Programming education research	Novice programmers	Debugging exercises	Error message structure	Error correction rates	Plain-language compiler messages improve debugging performance.	Error message readability barriers
Litherland, 2024	CS education discourse study	University CS students	Programming reasoning tasks	Linguistic discourse patterns	Communication patterns	Programming reasoning resembles conversational linguistic processes.	Programming as a language-mediated cognitive activity
Walker & Schach, 1996	Computer science education	Students learning programming languages	Programming assignments	Second-language learning effects	Programming performance	Students learning programming as a second language experience additional comprehension barrier.	Challenges faced by ESL (English as a second language) learners in computer science
Hao et al., 2019	Polyglot programming research	Professional developers	Code comprehension	Linguistic transfer skills	Cognitive strategies	Developers who speak multiple natural languages apply linguistic transfer skills in programming.	Programming as a language-mediated cognitive activity
Berssanette & Francisco, 2022	Computing education research	Undergraduate programming students	Learning tasks	Cognitive load and linguistic complexity	Cognitive load measures	Language-heavy instructional materials increase cognitive load during programming learning.	Cognitive load and syntax interpretation

Table 1, Figure 1

- Thematic analysis using hybrid inductive –deductive coding.
- Codes grouped into thematic clusters:
    - "Mental translation,"
    - "Lexical confusion,"
    - "Syntax interference,"
    - "English dependent debugging"
- Integrated Synthesis

Theme	Description
Language comprehension in programming education	Programming languages rely heavily on English vocabulary, keywords, and documentation. Learners must interpret linguistic structures embedded in programming syntax, identifiers, and instructions, making reading comprehension an important factor in programming success.
Cognitive load and syntax interpretation	Linguistic complexity in programming explanations and documentation can increase extraneous cognitive load, particularly for multilingual learners who must interpret or mentally translate technical English terminology while solving programming problems.
Error message readability barriers	Compiler error messages often contain dense technical language, and unfamiliar terminology can make debugging difficult for novice programmers. Simplifying error messages can improve comprehension and debugging accuracy.
Challenges faced by ESL (English as a second language) learners in computer science	Students who learn programming in a second language often experience additional barriers such as technical vocabulary comprehension, language anxiety, and difficulty interpreting instructions or documentation. These barriers can slow programming performance and increase cognitive effort.
Programming as a language-mediated cognitive activity	Research suggests programming comprehension involves cognitive processes like natural language processing. Understanding code requires linguistic interpretation alongside logical reasoning, indicating programming is partially a language-based activity.

Table Themes. Summary of the major themes identified in the systematic literature review examining the role of English language proficiency and linguistic factors in programming comprehension and problem-solving.

These themes form the basis of the final discussion and implications

Ethical Considerations

Because this study analyzes published research, no human subjects were involved. Ethical considerations include:

- Accurate reporting
- Transparent inclusion/exclusion
- Proper citation
- Avoiding manipulation of the result

Results

The systematic review revealed five major themes demonstrating how English language proficiency and linguistic factors shape programming problem solving. The themes highlighted consistent patterns across qualitative studies, showing language skills significantly influenced comprehension, error message debugging, and cognitive processing in programming environments.

Theme 1: Language Comprehension in Programming Education

Programming languages rely heavily on English vocabulary and grammatical structures. Keywords such as if, return, while, and case mirror natural language constructs and require learners to interpret linguistic meaning within a computational context. Research demonstrates students with stronger proficiency exhibit greater success in programming comprehension and problem solving. Identifier naming conventions greatly affects how much effort is needed to understand code (Binkley et al., 2013). Equally, English proficiency strongly predicts programming performance among software engineering students (Alrasheed, 2021).

Theme 2: Cognitive Load and Syntax Interpretation
Studies grounded in Cognitive Load Theory demonstrate programming tasks require learners to manage multiple cognitive processes simultaneously. When programming explanations contain linguistically complex instructions, multilingual learners must divide attention between language interpretation and logical reasoning. This process increases extraneous cognitive load and reduces available working memory resources (Mayer & Moreno, 2003; Sweller, 1988). Research by Durán et al. (2022) further signified English-heavy documentation increases cognitive strain and slows comprehension among novice programmers.

Theme 3: Error Message Readability Barriers
Compiler error messages represent one of the most common obstacles for programmers. These messages frequently contain technical terminology, abstract grammar, and unfamiliar vocabulary established that the students often misinterpret error messages due to their complexity (Becker et al., 2019). When error messages are rewritten in simplified language, debugging accuracy and task completion rates significantly improve. These findings highlight the importance of designing programming tools to present information in linguistically accessible ways.

Theme 4: Challenges Faced by ESL (English as a second language) Learners in Computer Science
English-as-a-second-language learners often face additional barriers when programming instruction assumes high levels of English proficiency. These barriers include difficulty interpreting technical vocabulary, increased anxiety during programming tasks, and challenges understanding assignment instructions. Research fortified linguistically declined students may spend additional cognitive effort translating programming terminology before reasoning about the problem itself (Alaofi & Russell, 2022). These linguistic barriers can slow learning progress and affect participation in computing programs.

Theme 5: Programming as a Language Mediated Cognitive Activity
The current recent research suggests the level of programming comprehension activates cognitive processes like those involved in natural language processing (Durán et al., 2022). Neurocognitive studies revealed understanding code engages brain regions associated with language comprehension and reasoning (Ivanova et al., 2020). This evidence supports the idea programming is not purely a mathematical activity but rather a hybrid cognitive process involving both linguistic interpretation and logical reasoning.

Discussion

The purpose of this systematic literature review remains to examine how English language proficiency and linguistic factors influence programming comprehension and problem-solving in computer science education. The synthesis of the selected studies reveals five primary themes: language comprehension in programming education, cognitive load and syntax interpretation, error message readability barriers, challenges faced by ESL (English as a second language) learners in computer science, and programming as a language-mediated cognitive activity. Together, these themes exhibit programming tasks require both logical reasoning and linguistic interpretation. The findings challenge the common perception of programming is purely a technical activity and instead suggest that programming comprehension is strongly influenced by language processing mechanisms.

The first theme, language comprehension in programming education, highlights the extent to which programming languages rely on English vocabulary and grammatical structures. Programming syntax, identifiers, documentation, and error messages are typically written in English, requiring learners to interpret linguistic meaning while simultaneously reasoning about program logic. Research reveals identifier naming conventions and textual clarity significantly influence programming comprehension and the effort required to interpret code (Binkley et al., 2013). Similarly, studies examining programming

education environments exhibited pupils with stronger English proficiency. These students tended to perform better on programming tasks. This required interpreting instructions and understanding documentation (Alrasheed, 2021). These findings hinted that programming comprehension involves reading and linguistic interpretation skills in addition to logical reasoning.

The second theme, cognitive load and syntax interpretation, emphasizes the cognitive demands associated with processing programming language structures. Cognitive Load Theory expounds learners have limited working memory resources available when solving complex tasks (Sweller, 1988). When programming materials contain dense technical language or unfamiliar terminology, learners may expend additional cognitive effort interpreting language rather than focusing on algorithmic reasoning. Research appears for programming documentation, and explanations rely heavily on complex English structures increase extraneous cognitive load (Durán et al., 2022). When cognitive resources are divided between language interpretation and problem solving, programming performance may be negatively affected. These findings support the importance of designing programming materials to minimize unnecessary linguistic complexity.

The third theme identified in this review concerns error message readability barriers. Debugging is a fundamental component of programming, yet compiler error messages often contain dense technical vocabulary and complex grammatical structures. Studies show that novice programmers frequently misinterpret error messages because they contain terminology or ambiguous phrasing (Becker et al., 2019). When error messages are rewritten using simpler and more accessible language, students show improved debugging accuracy and faster problem resolution. These findings denote improving the clarity and readability of compiler feedback could significantly enhance programming learning outcomes and reduce frustration among novice programmers.

The fourth theme addresses the challenges faced by ESL (English as a second language) learners in computer science education. Programming courses are frequently delivered using English-language instructional materials, documentation, and programming environments. For students with lower levels of English proficiency, interpreting these materials may require additional mental translation and vocabulary interpretation. Research point toward non-linguistic learners may spend additional cognitive effort translating programming terminology before engaging in logical tasks (Alaofi & Russell, 2022). This additional cognitive burden can slow learning progress and increase anxiety in programming environments. These findings highlight the need for instructional strategies to acknowledge linguistic diversity and provide support for multilingual learners in computing education.

The final theme in this review conceptualizes programming as a language mediated cognitive activity. Recent interdisciplinary research advocates code comprehension activates cognitive processes like those involved in natural language processing. Neuroimaging studies demonstrate programming tasks engage brain regions associated with language comprehension and executive reasoning (Ivanova et al., 2020). This evidence reinforces the idea programming involves interpretive processes like reading and language comprehension. Rather than viewing programming purely as mathematical logic, it may be more accurate to conceptualize programming as a hybrid activity that combines linguistic interpretation with computational reasoning.

Taken together, the themes in this systematic review suggest linguistic proficiency plays an important role in programming comprehension and problem-solving. Programming environments and instructional materials are often designed with implicit assumptions about English language proficiency, which may unintentionally disadvantage multilingual learners. Recognizing the linguistic dimensions of programming can help educators and software designers develop more inclusive instructional practices. Reducing unnecessary language complexity, improving error message clarity, and providing explicit instruction in

programming terminology may help reduce cognitive barriers and support more equitable learning environments in computer science education.

Thus, the findings contribute to a growing body of research views programming as a cognitively complex activity influenced by both logical reasoning and language processing. Understanding the role of linguistic factors in programming education may help researchers, educators, and software developers design more effective learning environments support diverse populations of learners entering the computing field.

Implications of Findings

The findings of this systematic literature review provide important insights into how English language proficiency influences programming comprehension, debugging, and problem-solving. Across the analyzed studies, programming tasks consistently required learners to interpret English-based keywords, documentation, and compiler messages. These linguistic demands create an additional layer of cognitive processing can affect programming performance, particularly for multilingual learners. Research in computing education determines identifier naming conventions, documentation clarity, and technical vocabulary significantly influence the effort required to interpret code (Binkley et al., 2013). As a result, English proficiency functions as a mediating factor can shape programming comprehension and learning outcomes.

The review also stressed the role of linguistic complexity in increasing cognitive load during programming tasks (Mayer & Moreno, 2003). When programming materials contain dense technical language, multilingual learners may need to mentally translate terminology before reasoning about program logic. This additional processing increases extraneous cognitive load and reduces the cognitive resources available for problem solving (Mayer & Moreno, 2003). Evidence from computing education research hints at simplifying error messages, documentation, and instructional materials can significantly improve debugging success and comprehension among novice programmers (Becker et al., 2019). These findings advise programming education, and development tools should consider linguistic accessibility as a key design factor.

Another implication of the findings concerns the broader accessibility and inclusivity of computer science education. Many programming environments and educational resources assume high levels of English proficiency. Studies examining English language learners in computer science programs direct language barriers can affect students' confidence, participation, and performance in courses (Lei et al., 2022). Addressing these challenges may require the development of linguistically inclusive instructional strategies, such as simplified programming documentation, clearer error messages, and explicit instruction in programming terminology. Improving linguistic accessibility could help reduce cognitive barriers and support a more diverse population of learners in computing fields.

Conclusion

This systematic literature review examined the role of English language proficiency and linguistic factors in programming problem-solving and comprehension. The synthesis of the selected studies establishes programming is not purely a logical or technical activity but also involves substantial linguistic processing. Programming languages rely heavily on English-based keywords, documentation, and syntax structures, requiring learners to interpret linguistic information while simultaneously applying logical reasoning. Evidence from the literature indicates factors such as identifier naming conventions, error message readability, and documentation (Binkley et al., 2013). In addition, clarity significantly influences programming comprehension and debugging performance (Becker et al., 2019).

The thematic synthesis of the literature identified five major themes: language comprehension in programming education, cognitive load and syntax interpretation, error message readability barriers, challenges faced by ESL (English as a second language) learners in computer science, and programming as a language-mediated cognitive activity. Together, these themes illustrate how linguistic proficiency interacts with cognitive processes during programming tasks. Learners with lower English proficiency may experience additional cognitive effort when interpreting programming materials, which can slow problem solving and increase frustration during debugging activities.

Overall, the findings suggest English proficiency functions as an important mediator in programming performance. Recognizing the linguistic dimensions of programming can help educators and software designers develop more inclusive instructional practices and tools. By reducing unnecessary linguistic complexity and improving the clarity of programming resources, it may be possible to create learning environments better support diverse learners in computer science education.

The influence of English language proficiency in programming learners' comprehension, problem solving processes, and overall learning outcomes in computing education contexts is answered by the English language proficiency has a measurable impact on programming learners' comprehension, problem solving ability, and overall academic performance in computing education. Programming languages, documentation, and instructional resources are largely written in English. Learners must interpret both linguistic meaning and computational logic at the same time. Research indicates students with stronger English proficiency demonstrate improved understanding of programming syntax, semantic structures, and algorithmic reasoning processes. Programming constructs often reflect natural language patterns, requiring learners to decode vocabulary and contextual meaning before applying logical problem-solving strategies. Studies also present comprehension of identifier names and technical terminology influences how efficiently learners read and interpret code. Additionally, neurocognitive research suggests programming engages language related cognitive processes, reinforcing the idea of programming involves both linguistic interpretation and logical reasoning skills. These findings show that English proficiency plays a central role in shaping programming comprehension and learning outcomes.

The linguistic, cognitive, and instructional barriers are documented in the literature regarding linguistic learners' experiences in programming and computer science education is identified in the literature identifies several linguistic, cognitive, and instructional barriers English as an additional language learners experience in computer science education. Cognitive Load Theory explains that learners must divide their attention between understanding English language explanations and applying computational thinking. Which increases extraneous cognitive load and reduces working memory capacity available for problem solving tasks. English intensive instructional materials and technical documentation can slow comprehension because learners may need to mentally translate unfamiliar vocabulary before interpreting programming concepts. Research also shows the compiler error messages frequently include complex grammatical structures and specialized terminology that can be difficult for multilingual learners to interpret. This can lead to misunderstandings and reduced debugging effectiveness. Additional barriers include language related anxiety, difficulty interpreting assignment instructions, and increased time needed to understand domain specific terminology used in programming education.

Evidence-based instructional strategies, support mechanisms, and learning environment practices are recommended across studies to enhance linguistic inclusivity in programming education by the research
Provided by evidence-based recommendations for improving linguistic inclusivity in programming education through supportive instructional strategies and learning environment design. Suggested approaches include simplifying technical explanations, improving clarity of error messages, and providing structured learning reinforces reduced language related cognitive load. Resources such as glossaries of programming terminology, visual learning support, and instructional materials can assist learners in focusing on logical reasoning rather than language translation challenges. Collaborative learning

environments and accessible instructional design practices further support comprehension and participation among non-adaptive linguistic learners. These strategies promote more inclusive computing education environments, improve problem solving performance, and help ensure equitable learning opportunities for students from diverse linguistic backgrounds.

Limitations of the Study

Several limitations should be considered when interpreting the findings of this systematic literature review. First, the review focused primarily on peer-reviewed studies published in English-language academic databases. This restriction may have excluded relevant research conducted in other languages or published in regional journals that were not indexed in the selected databases. Consequently, the results may not fully represent the global diversity of programming education research.

A second limitation involves the variability of research methods used across the included studies. The selected literature employed a wide range of methodologies, including experimental studies, qualitative analyses, neuroimaging research, and systematic reviews. These methodological differences make direct comparisons across studies more difficult and may limit the ability to generalize findings across programming education contexts. Additionally, many of the studies focused on undergraduate students or novice programmers in academic environments, which may not fully reflect the experiences of professional developers working in multilingual software development teams.

Next, because this study is based on a systematic review of existing literature, the findings are dependent on the scope and quality of previously published research. Some studies lacked standardized measures of English proficiency or programming performance, which may affect the consistency of the evidence base. Future research using standardized measurement approaches could provide more precise insights.

Recommendations for Future Research

Future research should continue exploring the relationship between language proficiency and programming performance through empirical and experimental approaches. One promising direction involves the use of cognitive measurement techniques such as eye tracking, cognitive load assessment, and neuroimaging to examine how learners process programming languages and documentation in real time. Previous studies have exhibited programming comprehension activates cognitive processes like those used in natural language processing, suggesting linguistic mechanisms play a meaningful role in programming tasks (Ivanova et al., 2020).

Lastly, additional research is also needed to evaluate instructional interventions designed to reduce linguistic barriers in programming education. For example, studies could investigate the effectiveness of simplified compiler error messages, multilingual documentation, or programming curricula overtly teach programming vocabulary and terminology. Controlled experiments comparing traditional programming materials with linguistically simplified versions could provide valuable evidence about how instructional design influences programming comprehension and debugging success. Beyond doubt, future research should examine the long-term effects of linguistic barriers on participation and persistence in computer science education. Longitudinal studies tracking multilingual learners throughout programming curricula could help determine how language proficiency influences skill development, confidence, and career pathways in computing fields. Expanding research in these areas may help educators and software developers design more inclusive learning environments that support a broader and more diverse population of programmers.

References

Alaofi, S., & Russell, S. (2022). A study of foreign language classroom anxiety and its effect on academic performance in English-based CS1 courses. *In Proceedings of the 24th Australasian Computing Education Conference (ACE 2022)*. https://doi.org/10.1145/3555009.3555020

Alrasheed, H. (2021). Impact of English proficiency on academic performance of software engineering students. *In Proceedings of the 3rd International Conference on Education and E-Learning*. https://doi.org/10.1145/3456146.3456163

Avidan, E., & Feitelson, D. G. (2017). Effects of variable names on comprehension: An empirical study. *2017 IEEE 25th International Conference on Program Comprehension (ICPC)*. https://doi.org/10.1109/ICPC.2017.27

Berssanette, J. H., & de Francisco, A. C. (2022). Cognitive load theory in the context of teaching and learning computer programming: A systematic literature review. *IEEE Transactions on Education, 65(3), 440–449*. https://doi.org/10.1109/TE.2021.3127215

Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., & Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empirical Software Engineering, 18(2), 219–276*. https://doi.org/10.1007/s10664-012-9201-4

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.-M., Pearce, J. L., & Prather, J. (2019). Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *In Proceedings of the 2019 ITiCSE Working Group Reports (ITiCSE-WGR '19)* (pp. 177–210). ACM. https://doi.org/10.1145/3344429.3372508

Becker, B. A., Glanville, G., Iwashima, R., McDonnell, C., Goslin, K., & Mooney, C. (2016). Effective compiler error message enhancement for novice programming students. *Computer Science Education, 26(*2–3), 148–175. https://doi.org/10.1080/08993408.2016.1225464

Boroditsky, L. (2011). How language shapes thought. *Scientific American, 304(2), 62–65*. https://doi.org/10.1038/scientificamerican0211-62

Duran, R. S., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive load theory in computing education research: A review. *ACM Transactions on Computing Education, 22*(4), Article 40, 1–27. https://doi.org/10.1145/3483843

Gordon, C.S., (2024). The linguistics of programming. In Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '24) (pp. 162–182). *Association for Computing Machinery*. https://doi.org/10.1145/3689492.3689806

Groeneveld, W., Becker, B. A., & Vennekens, J. (2022). *How creatively are we teaching and assessing creativity in computing education: A systematic literature review*. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education (SIGCSE 2022). https://doi.org/10.1145/3478431.3499360

Hao, R. & Glassman , E. (2019). Approaching polyglot programming: approaching polyglot programming: what can we learn from bilingualism ttudies? *Oasis PLATEAU 2019, Article 1.* https://doi.org/10.4230/OASIcs.PLATEAU.2019.1

Hermans, F., (2020). Hedy: A Gradual Language for Programming Education. *Proceedings of the ACM on Human-Computer Interaction, 4(ISSE), 1–17*. https://doi.org/10.1145/3372782.3406262

Ivanova A., Srikant S., Sueoka Y., Kean H., Dhamala. R, O'Reilly, U., Marina. B., Fedorenko,E. (2020) Comprehension of computer code relies primarily on domain-general executive brain regions eLife https://doi.org/10.7554/eLife.58906

Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (*EBSE Technical Report EBSE-2007-01*). Keele University & Durham University. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf

Lorås, M., Sindre, G., Trætteberg, H., & Aalberg, T. (2021). *Study behavior in computing education: A systematic literature review*. ACM Transactions on Computing Education, 22(1), Article 1. https://doi.org/10.1145/3469129

Lei, Y., & Allen, M. (2022). *English language learners in computer science education: A scoping review. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education (SIGCSE '22) (pp. 57–63). Association for Computing Machinery*. https://doi.org/10.1145/3478431.3499299

Litherland, K. T. (2024). Learning to program as empirical inquiry: Using a conversation perspective to explore student programming processes. *Computer Science Education. Advance online publication*. https://doi.org/10.1080/08993408.2023.2290410

Mayer, R. E., & Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. *Educational Psychologist, 38(1), 43–52*. https://doi.org/10.1207/S15326985EP3801_6

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. BMJ, 372, n71. https://doi.org/10.1136/bmj.n71

Siegmund, J., Peitek, N., Parnin, C., Apel, S., Hofmeister, J., Kästner, C., Begel, A., Bethmann, A., & Brechmann, A. (2017). Measuring neural efficiency of program comprehension. *In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017) (pp. 140–150). Association for Computing Machinery*. https://doi.org/10.1145/3106237.3106268

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12*(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4

Sweller, J., van Merriënboer, J. J. G., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review, 31*(2*), 261–292*. https://doi.org/10.1007/s10648-019-09465-5

Walker, K. P., & Schach, S. R. (1996). Obstacles to learning a second programming language: An empirical study. *Computer Science Education*, *7*(1), *1–20*. https://doi.org/10.1080/0899340960070101