Revista Crátilo, vol. 18, n. 2: 90-103, jul./dez. 2025
© Centro Universitário de Patos de Minas
https://revistas.unipam.edu.br/index.php/cratilo

A Programação como Linguística Aplicada: uma abordagem epistemológica e teórica

Programming as Applied Linguistics: an Epistemological and Theoretical Approach

ISABELLA TAVARES SOZZA MORAES
Mestre em Letras (USP)
sozzaisabella@gmail.com

Resumo: Este artigo investigou a natureza epistemológica da programação de computadores, argumentando que sua essência reside mais no campo da linguística aplicada do que no domínio estrito da tecnologia. Partindo de uma revisão teórica fundamentada em autores como Chomsky (1957), Knuth (1968) e Saussure (1916), o estudo evidenciou as estruturas sintáticas, semânticas e pragmáticas das linguagens de programação, analisando-as como sistemas de signos que medeiam a comunicação entre seres humanos e máquinas. A metodologia adotada foi de natureza qualitativa e teórico-conceitual, baseada em análise bibliográfica de fontes primárias e secundárias. O problema central residiu na desconstrução da visão instrumentalista que reduz a programação a uma mera ferramenta tecnológica, propondo, em contrapartida, uma compreensão mais ampla e humanística. Os resultados indicaram que a programação compartilha fundamentos com a linguística teórica, especialmente no que tange à gramática generativa, à análise de discurso e à pragmática. Concluiu-se que reposicionar a programação no campo da linguística aplicada pode enriquecer seu ensino, prática e desenvolvimento teórico, além de fomentar diálogos interdisciplinares entre ciência da computação, filosofia da linguagem e ciências humanas.
Palavras-chave: Programação; Linguística Aplicada; Epistemologia; Sintaxe Computacional; Semântica Formal.

Abstract: This article investigated the epistemological nature of computer programming, arguing that its essence lies more within the field of applied linguistics than within the strictly technological domain. Drawing on a theoretical review grounded in authors such as Chomsky (1957), Knuth (1968), and Saussure (1916), the study highlighted the syntactic, semantic, and pragmatic structures of programming languages, analyzing them as sign systems that mediate communication between humans and machines. The methodology adopted was qualitative and theoretical-conceptual, based on a bibliographic analysis of primary and secondary sources. The central problem involved deconstructing the instrumentalist view that reduces programming to a mere technological tool, proposing instead a broader and more humanistic understanding. The results indicated that programming shares foundational principles with theoretical linguistics, particularly with regard to generative grammar, discourse analysis, and pragmatics. It was concluded that repositioning programming within the field of applied linguistics can enrich its teaching, practice, and theoretical development, as well as foster interdisciplinary dialogues among computer science, philosophy of language, and the humanities.
Keywords: Programming; Applied Linguistics; Epistemology; Computational Syntax; Formal Semantics.

1 INTRODUÇÃO

A programação de computadores tem sido tradicionalmente enquadrada como uma disciplina técnica, inserida no âmbito das ciências exatas e da engenharia. No entanto, uma análise mais aprofundada de sua natureza revela que seus fundamentos estão mais alinhados com a Linguística Aplicada do que com a tecnologia pura.

Esta pesquisa busca deslocar o eixo de compreensão da programação, reposicionando-a como uma prática linguística que opera por meio de sistemas simbólicos estruturados, os quais permitem a comunicação entre programadores e máquinas. A justificativa para tal abordagem reside na necessidade de superar visões reducionistas que limitam o potencial criativo e expressivo inerente à atividade de programar, além de fomentar um diálogo interdisciplinar que una ciência da computação, filosofia da linguagem e estudos da comunicação.

O arcabouço teórico que sustenta esta investigação evidencia contribuições seminais de Noam Chomsky, cuja teoria da gramática generativa torna em evidência um paralelo robusto entre a estrutura das linguagens naturais e a das linguagens de programação (Chomsky, 1957). Da mesma forma, os trabalhos de Donald Knuth sobre a 'arte de programar' destacam a dimensão estética e literária do código-fonte, aproximando-o de uma forma de expressão escrita (Knuth, 1984). A noção de signo linguístico, desenvolvida por Ferdinand de Saussure, também é fundamental para compreender como os símbolos e comandos em programação adquirem significado dentro de um sistema fechado de regras (Saussure, 1916).

A metodologia empregada é de caráter qualitativo e teórico-conceitual, baseando-se na análise crítica de bibliografia especializada e na interpretação de exemplos concretos de linguagens de programação, como *Python*, *Haskell e Ves*, esta última, uma linguagem recentemente proposta por Silva (2019) em seu trabalho de conclusão de curso. A escolha por uma abordagem teórica justifica-se pela natureza epistemológica da questão investigada, que demanda reflexão conceptual e articulação entre diferentes campos do saber.

O objetivo geral deste artigo é demonstrar que a programação pode ser compreendida como uma forma de linguística aplicada, na medida em que envolve a criação, manipulação e interpretação de sistemas simbólicos com regras sintáticas e semânticas definidas. Como objetivos específicos, busca-se: analisar as estruturas gramaticais das linguagens de programação à luz da linguística chomskiana; explorar a dimensão pragmática do código como ato de comunicação; e discutir as implicações pedagógicas e epistemológicas desse reposicionamento teórico.

O problema de pesquisa pode ser formulado da seguinte maneira: em que medida a programação de computadores pode ser redefinida como uma prática de linguística aplicada e quais as consequências desse reposicionamento para seu ensino, desenvolvimento teórico e compreensão social? A hipótese central é que a programação, longe de ser uma mera ferramenta tecnológica, constitui-se como uma linguagem formal que compartilha fundamentos com as línguas humanas, podendo ser estudada e praticada a partir de referenciais teóricos da linguística.

Ao longo deste artigo, serão desenvolvidos três capítulos principais. O primeiro dedicar-se-á à fundamentação teórica inicial, apresentando os conceitos-chave da

linguística que se aplicam à programação. O segundo capítulo aprofundará a discussão científica, analisando casos concretos e dialogando com autores contemporâneos. O terceiro capítulo apresentará os resultados e interpretações, articulando as evidências coletadas com a hipótese proposta. Por fim, as considerações finais sintetizarão as contribuições do estudo e apontarão direções para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA INICIAL

A interface entre linguística e programação de computadores remonta aos primórdios da ciência da computação, quando pioneiros como Alan Turing e John von Neumann já reconheciam a importância de sistemas de símbolos para a comunicação com máquinas. No entanto, foi com o advento das linguagens de alto nível, como Fortran e Lisp, que a analogia com as línguas humanas se tornou mais evidente. Nesse sentido, a teoria linguística de Noam Chomsky torna em evidência um ponto de partida sólido para compreender a estrutura formal das linguagens de programação. Chomsky (1957) propôs uma hierarquia de gramáticas formais que classifica as linguagens de acordo com sua complexidade sintática, categorização que foi diretamente aplicada no design de compiladores e analisadores sintáticos. A gramática livre de contexto, por exemplo, é amplamente utilizada na especificação de linguagens de programação, como observa Aho *et al*. (2006) em seu clássico *Compilers: Principles, Techniques, and Tools.*

Além da sintaxe, a semântica das linguagens de programação também pode ser estudada à luz de teorias linguísticas. A semântica formal, tal como desenvolvida por Richard Montague (1970), fornece ferramentas para analisar o significado de construções linguísticas por meio da lógica matemática. De modo análogo, a semântica das linguagens de programação preocupa-se com a interpretação dos comandos e expressões, definindo como estes são executados pela máquina. Winskel (1993) explora essa relação em *The Formal Semantics of Programming Languages*, demonstrando que conceitos como denotação, operação e axiomas semânticos têm correspondentes diretos na filosofia da linguagem.

Outro conceito fundamental é o de pragmática, que na linguística estuda o uso da linguagem em contextos específicos e os efeitos desse uso na comunicação. Na programação, a pragmática manifesta-se na legibilidade, estilo e intenção do código, aspectos que Donald Knuth (1984) aborda em *Literate Programming.* Para Knuth, programar é uma atividade literária, na qual o programador deve escrever códigos que sejam compreensíveis não apenas para a máquina, mas também para outros seres humanos. Essa perspectiva ressalta a dimensão social e comunicativa da programação, alinhando-a com os estudos de pragmática linguística desenvolvidos por autores como J. L. Austin (1962) e John Searle (1969), que investigam os atos de fala e a intencionalidade por trás dos enunciados.

A noção de signo proposta por Ferdinand de Saussure (1916) também se mostra relevante para a análise das linguagens de programação. Para Saussure, o signo linguístico é composto por um significante (a forma) e um significado (o conceito). Na programação, os significantes são os *tokens* – identificadores, palavras-chave, operadores –, enquanto os significados são os conceitos computacionais que esses tokens representam, como variáveis, funções e estruturas de controle. No entanto, ao contrário

das línguas naturais, em que a relação entre significante e significado é arbitrária, nas linguagens de programação essa relação é convencional e rigidamente definida pela gramática da linguagem. Essa característica aproxima as linguagens de programação das línguas formais, como a lógica simbólica, mas não as desvincula completamente da linguística, uma vez que a criação e a evolução dessas linguagens envolvem escolhas humanas influenciadas por fatores históricos, culturais e ergonômicos.

A linguagem Ves, proposta por Silva (2019), serve como exemplo contemporâneo de como conceitos linguísticos são incorporados ao design de linguagens de programação. Ves prioriza a legibilidade e a expressividade, adotando uma sintaxe declarativa e suporte a pattern matching, características que refletem uma preocupação com a clareza comunicativa e a eficácia na transmissão de intenções. Ao analisar a gramática de Ves, é possível identificar estruturas que espelham construções de línguas naturais, como a utilização de cláusulas condicionais inline e a definição de funções por meio de correspondência de padrões, recursos estes que facilitam a leitura e a compreensão do código.

A fundamentação teórica inicial demonstra que a programação compartilha com a linguística uma série de conceitos e métodos, desde a análise sintática e semântica até a pragmática e a teoria do signo. Essas afinidades sugerem que a programação pode ser furtivamente estudada como uma forma de linguística aplicada, o que abre caminho para investigações mais aprofundadas sobre suas implicações epistemológicas e pedagógicas.

2.1 BASES LINGUÍSTICAS DA PROGRAMAÇÃO: DA GRAMÁTICA GENERATIVA ÀS LINGUAGENS FORMAIS

Como exposto na seção anterior, a interface entre linguística e programação foi estabelecida desde os primórdios da computação, evidenciando a centralidade dos sistemas simbólicos na comunicação entre humanos e máquinas. Retomando e aprofundando essa base teórica, interessa agora examinar de forma mais técnica como os princípios da gramática formal — especialmente a hierarquia proposta por Chomsky (1957) — estruturam diretamente o funcionamento das linguagens de programação e orientam o design de compiladores, analisadores sintáticos e gramáticas livres de contexto amplamente utilizadas no domínio computacional.

Além da sintaxe, a semântica das linguagens de programação também pode ser estudada à luz de teorias linguísticas. A semântica formal, tal como desenvolvida por Richard Montague (1970), fornece ferramentas para analisar o significado de construções linguísticas por meio da lógica matemática. De modo análogo, a semântica das linguagens de programação preocupa-se com a interpretação dos comandos e expressões, definindo como estes são executados pela máquina. Winskel (1993) explora essa relação em *The Formal Semantics of Programming Languages*, demonstrando que conceitos como denotação, operação e axiomas semânticos têm correspondentes diretos na filosofia da linguagem.

A noção de signo proposta por Ferdinand de Saussure (1916) também se mostra relevante para a análise das linguagens de programação. Para Saussure, o signo linguístico é composto por um significante (a forma) e um significado (o conceito). Essa

característica aproxima as linguagens de programação das línguas formais, como a lógica simbólica, mas não as desvincula completamente da linguística, uma vez que a criação e a evolução dessas linguagens envolvem escolhas humanas influenciadas por fatores históricos, culturais e ergonômicos.

O advento da inteligência artificial, particularmente dos modelos de processamento de linguagem natural (PLN), criou uma ponte substantiva entre a linguística teórica e a ciência da computação. Segundo Jurafsky e Martin (2021), o PLN representa a convergência entre teorias linguísticas formais e implementações computacionais, em que conceitos como sintaxe, semântica e pragmática são operacionalizados em algoritmos que processam linguagem humana. Esta intersecção demonstra que a programação não é meramente tecnologia, mas sim linguística aplicada em seu sentido mais amplo.

Os modelos de linguagem de grande escala, como os baseados na arquitetura *transformer*, evidenciam esta conexão de maneira particularmente relevante. Vaswani *et al*. (2017), em seu trabalho seminal *Attention Is All You Need*, demonstraram como mecanismos de atenção poderiam capturar relações sintáticas e semânticas complexas sem a necessidade de gramáticas formais explicitamente programadas. No entanto, como observa Bender *et al.* (2021), esses modelos ainda dependem fundamentalmente de insights da teoria linguística, particularmente na representação distribuída de significado e na estrutura hierárquica da linguagem.

A contribuição da linguística de *corpus* para o desenvolvimento de sistemas de IA further ilustra essa interligação. Segundo Manning (2015), a análise computacional de grandes corpora linguísticos não apenas permitiu o avanço dos sistemas de PLN, mas também forneceu novos insights para teorias linguísticas tradicionais. Através de técnicas como word embeddings e análise de dependências, os linguistas computacionais puderam validar empiricamente hipóteses sobre aquisição de linguagem, variação linguística e mudança diacrônica.

A linguagem Ves, analisada nesse contexto, exemplifica como princípios tanto da linguística tradicional quanto da computação moderna podem convergir no design de linguagens de programação. Silva (2019) priorizou a legibilidade e a expressividade, adotando uma sintaxe declarativa que facilita tanto a compreensão humana quanto o processamento automatizado.

Goldberg (2017) argumenta que a neurociência computacional e a linguística teórica estão convergindo para uma nova compreensão da linguagem como fenômeno tanto biológico quanto computacional. Essa perspectiva unificada sugere que a distinção entre programação e linguística é artificial, sendo mais produtivo entender ambas como facetas do estudo dos sistemas simbólicos e sua implementação em diferentes substratos, biológicos ou digitais.

A pragmática computacional, como explorado por Ginzburg (2012), torna em evidência outro exemplo dessa convergência. Modelos computacionais de diálogo e interação linguística requerem a formalização de conceitos pragmáticos como pressuposição, implicatura e atos de fala, demonstrando como teorias linguísticas abstratas encontram aplicação prática em sistemas de IA conversacional. A inteligência artificial também contribui para o desenvolvimento teórico da linguística, criando um ciclo virtuoso de influência mútua.

3 DESENVOLVIMENTO E DISCUSSÃO CIENTÍFICA

A análise das linguagens de programação sob a ótica da linguística aplicada permite explorar dimensões que frequentemente são negligenciadas pela abordagem puramente técnica. Um desses aspectos é a variação linguística, conceito central na sociolinguística que se refere às diferentes formas que uma língua pode assumir em função de fatores geográficos, sociais ou contextuais. Na programação, a variação manifesta-se na existência de múltiplas linguagens e paradigmas – como imperativo, funcional e orientado a objetos –, cada um com seu próprio vocabulário, sintaxe e estilo. Essa diversidade não é meramente técnica, mas reflete diferentes filosofias de design e comunidades de prática, tal como ocorre com os dialetos nas línguas humanas. Guido van Rossum (1997), criador do Python, justificou as escolhas de design de sua linguagem com base na legibilidade e na simplicidade, valores que ecoam preocupações típicas da linguística aplicada, como a adequação da linguagem ao contexto de uso e ao perfil dos falantes – ou, no caso, dos programadores.

Outro ponto de convergência entre programação e linguística é a aquisição da linguagem. Assim como as crianças aprendem a falar por meio da exposição a input linguístico e da prática comunicativa, os programadores aprendem a codificar por meio da imersão em exemplos de código, da leitura de documentação e da resolução de problemas. Pesquisas na área de educação em computação, como as conduzidas por Soloway e Spohrer (1989), mostram que a aprendizagem de programação envolve processos cognitivos similares aos da aquisição de uma segunda língua, evidenciando a internalização de regras gramaticais, o desenvolvimento de competência comunicativa e a transferência de conhecimentos entre linguagens.

A análise do discurso, campo da linguística que estuda a organização e o funcionamento dos textos em situações comunicativas, também se aplica à programação. O código-fonte pode ser entendido como um gênero discursivo específico, com convenções de estruturação, estilo e coerência. Estudos como os de Blackwell *et al.* (2014) investigam como programadores leem e interpretam códigos, identificando estratégias de leitura que se assemelham às utilizadas na compreensão de textos em língua natural. Por exemplo, a capacidade de inferir a intenção do autor a partir de pistas contextuais – como nomes de variáveis e comentários – é uma habilidade tanto linguística quanto computacional. Nesse sentido, a programação aproxima-se de atividades tradicionais de letramento, demandando competências de leitura e escrita que vão além do domínio técnico.

A linguagem Ves, analisada no capítulo anterior, ilustra como preocupações linguísticas influenciam o design de linguagens de programação. Silva (2019) enfatiza a importância da legibilidade e da expressividade, propondo uma sintaxe que minimize ambiguidades e maximize a clareza. Além disso, Ves incorpora recursos como *pattern matching* e uniões discriminadas, que permitem expressar algoritmos de forma mais concisa e natural, reduzindo a distância cognitiva entre o pensamento do programador e a representação codificada. Tais características reforçam a ideia de que a programação é, em essência, uma atividade de modelagem linguística.

No entanto, é importante reconhecer as limitações da analogia entre programação e linguística. Enquanto as línguas naturais evoluem organicamente e são

marcadas pela ambiguidade, pela criatividade e pela variação irrestrita, as linguagens de programação são sistemas formais, restritos e previsíveis. Essa diferença, porém, não invalida a aproximação proposta, dessa forma a complexifica, mostrando que a programação ocupa um espaço híbrido entre a linguística pura e a engenharia. Como argumenta Raymond (1999), em *The Cathedral and the Bazaar*, a cultura do *open source* exemplifica como comunidades de programadores desenvolvem dialetos próprios, negociam significados e estabelecem convenções de estilo, processos estes que são intrinsecamente linguísticos.

3.1 A PERSISTÊNCIA DOS FUNDAMENTOS SAUSSURIANOS NA ERA COMPUTACIONAL

A teoria linguística de Ferdinand de Saussure (1916), frequentemente considerada o marco fundador da linguística moderna, mantém relevância surpreendente na compreensão das linguagens de programação e dos sistemas de inteligência artificial contemporâneos. A distinção saussuriana entre *langue* (sistema linguístico) e *parole* (fala individual) encontra um paralelo notável na distinção entre a especificação formal de uma linguagem de programação e sua implementação em código executável. Como observa Silva (2019), em sua análise da linguagem Ves, a definição da gramática e dos construtores sintáticos representa a *langue*, enquanto os programas escritos nessa linguagem constituem a *parole* computacional.

O conceito de valor na teoria saussuriana adquire dimensões particulares no contexto da programação. Para Saussure (1916), o valor de um signo linguístico é determinado por suas relações diferenciais dentro do sistema, e não por qualquer conexão intrínseca com o mundo exterior. Na programação, este princípio manifesta-se na maneira como identificadores, tipos e operações adquirem significado através de suas posições na gramática da linguagem e no sistema de tipos. Como demonstram Jurafsky e Martin (2021), mesmo nos modelos de IA mais avançados, a representação de significado continua dependente de relações sistêmicas, embora estas sejam capturadas estatisticamente em vez de serem definidas explicitamente. A noção de arbitrariedade do signo, central na teoria saussuriana, também se aplica às escolhas de sintaxe em linguagens de programação, em que diferentes significantes podem representar os mesmos conceitos computacionais, conforme observado na comparação entre linguagens como Python, C++ e Ves realizada por Silva (2019).

A análise das unidades mínimas de significado, outro conceito fundamental da linguística estrutural, mostra-se igualmente relevante para a compreensão dos sistemas computacionais. Tal como os morfemas constituem as unidades mínimas portadoras de significado nas línguas naturais, os tokens e expressões primitivas nas linguagens de programação funcionam como blocos construtivos elementares. No entanto, como argumenta Manning (2015), os modelos de IA contemporâneos desafiam esta visão discreta ao operarem com representações contínuas e distribuídas do significado. Essa tensão entre representações discretas e contínuas reflete um debate mais amplo na linguística teórica, que remonta às críticas pós-estruturalistas ao modelo saussuriano.

A evolução da linguística teórica após Saussure trouxe contribuições essenciais para a compreensão da programação como atividade linguística. A teoria dos atos de

fala de Austin (1962) e Searle (1969), originalmente desenvolvida para analisar a linguagem natural, mostrou-se útil para compreender a pragmática das linguagens de programação. Como observa Blackwell *et al*. (2014), os comandos de programação podem ser entendidos como atos ilocucionários que não apenas descrevem estados computacionais, mas também os alteram através de sua execução. Essa perspectiva permite analisar a programação como uma forma de ação linguística, em que a escrita de código constitui performance e não apenas descrição.

A gramática sistêmico-funcional de Halliday (1985), com seu foco na relação entre estrutura linguística e função social, torna em evidência outra lente valiosa para analisar as linguagens de programação. Segundo essa abordagem, as escolhas sintáticas em programação refletem não apenas considerações técnicas, mas também contextos sociais e comunicativos específicos. A linguagem Ves, por exemplo, com sua ênfase na legibilidade e expressividade, reflete um contexto em que a comunicação entre programadores é valorizada tanto quanto a eficiência computacional. Como demonstra Silva (2019), características como a sintaxe declarativa e o pattern matching facilitam a leitura e compreensão do código, priorizando assim a função interpessoal da linguagem.

A linguística cognitiva, particularmente a teoria dos protótipos de Rosch (1975) e a gramática de construções de Goldberg (1995), fornece insights importantes para entender como os programadores categorizam e organizam conceitos computacionais. Estudos como os de Medeiros *et al*. (2020) mostram que aprendizes de programação frequentemente dependem de categorização baseada em protótipos para entender conceitos como "objeto", "função" ou "classe". Essa perspectiva cognitiva ajuda a explicar por que algumas construções sintáticas são mais intuitivas que outras e por que linguagens como Python e Ves, que se alinham mais closely com categorizações cognitivas naturais, tendem a ser consideradas mais acessíveis para iniciantes.

A teoria da relevância de Sperber e Wilson (1986), originalmente desenvolvida para explicar a compreensão na comunicação humana, mostrou-se aplicável também à análise de como programadores interpretam código. Segundo essa abordagem, a compreensão de um programa envolve inferências baseadas no princípio da relevância, em que o leitor busca maximizar os efeitos cognitivos enquanto minimiza o esforço de processamento. Isto explica por que características como nomes significativos de variáveis, comentários bem colocados e estruturação clara do código – todas presentes na linguagem Ves conforme descrita por Silva (2019) – facilitam significativamente a compreensão.

A convergência entre teorias linguísticas contemporâneas e desenvolvimento de sistemas de IA further enriquece este diálogo interdisciplinar. Como observa Bender *et al*. (2021), os desafios enfrentados no desenvolvimento de sistemas de processamento de linguagem natural frequentemente ressaltam a importância de insights da linguística teórica. A tensão entre abordagens baseadas em *corpus* e abordagens baseadas em regras espelha debates mais amplos na linguística entre descrição e prescrição, entre uso real e sistema abstrato.

4 RESULTADOS E INTERPRETAÇÕES

A análise teórica e conceitual desenvolvida nos capítulos anteriores permitiu identificar uma série de evidências que sustentam a hipótese de que a programação pode ser compreendida como linguística aplicada. Uma dessas evidências é a centralidade da sintaxe e da semântica tanto na linguística quanto na ciência da computação. Como demonstrado por Chomsky (1957) e por Aho *et al*. (2006), a especificação formal de linguagens – naturais ou de programação – requer a definição precisa de regras gramaticais e de interpretação. Na prática, isso significa que programadores, assim como linguistas, trabalham com sistemas de regras que determinam a estrutura e o significado dos enunciados. A linguagem Ves, por exemplo, possui uma gramática explicitamente descrita por Silva (2019) usando a ferramenta ANTLR, o que permite visualizar as produções sintáticas de modo similar à análise de frases em língua natural.

Outro resultado significativo diz respeito à pragmática do código. Observou-se que a legibilidade, a expressividade e o estilo são valores cultivados por comunidades de programação, tal como a clareza e a coerência são valorizadas em textos escritos. Knuth (1984) defende que programar é uma atividade literária, na qual o autor deve preocupar-se não apenas com a correção funcional, mas também com a elegância e a comunicabilidade do código. Essa visão é compartilhada por autores como Martin (2008), que, em *Clean Code*, argumenta que código bem escrito é fácil de ler, modificar e manter, qualidades estas que dependem de escolhas linguísticas conscientes. Tais preocupações refletem uma dimensão pragmática da programação, alinhando-a com os estudos de uso da linguagem em contextos reais.

A noção de aquisição de linguagem também se mostrou relevante para a compreensão do aprendizado de programação. Estudos como os de Medeiros *et al*. (2020) indicam que a dificuldade inicial de muitos estudantes em programar está relacionada à incapacidade de internalizar a sintaxe e a semântica da linguagem, problema análogo ao enfrentado por aprendizes de línguas estrangeiras. Isso sugere que estratégias pedagógicas baseadas em imersão, prática repetida e feedback contextualizado – comuns no ensino de línguas – podem ser eficazes no ensino de programação. Além disso, a transferência de conhecimentos entre linguagens de programação assemelha-se à transferência interlinguística observada em bilíngues, fenômeno estudado pela psicolinguística.

A análise da linguagem Ves corroborou a ideia de que o design de linguagens de programação é influenciado por critérios linguísticos. Silva (2019) priorizou a simplicidade e a legibilidade, adotando uma sintaxe declarativa e suporte a construções como match e loop nomeados, que aproximam o código da lógica humana. Essas escolhas refletem uma preocupação com a eficácia comunicativa, valor central da linguística aplicada. Ademais, a inclusão de comentários e a definição de identificadores claros em Ves reforçam a ideia de que o código é um texto destinado tanto a humanos quanto a máquinas, devendo, portanto, ser pensado como um produto de comunicação.

A investigação revelou que a distinção entre programação e linguística é, em grande medida, artificial e historicamente construída. Enquanto a ciência da computação emergiu como disciplina autônoma no século XX, seus fundamentos linguísticos remontam à lógica aristotélica e à filosofia da linguagem. Reconhecer essa continuidade

histórica permite resgatar a dimensão humanística da programação, frequentemente obscurecida pelo discurso tecnocrático. Como argumenta Dijck (2013), a cultura digital contemporânea demanda cidadãos que sejam não apenas consumidores de tecnologia, mas também produtores críticos de significado, competência esta que depende do letramento computational entendido como uma forma de letramento linguístico.

Assim, os resultados obtidos sustentam a hipótese de que a programação pode ser redefinida como linguística aplicada, desde que se adote uma perspectiva ampla e inclusiva do que constitui a linguagem.

4.1 ANÁLISE LINGUÍSTICA DE CONSTRUCTOS SINTÁTICOS EM LINGUAGENS DE PROGRAMAÇÃO

A análise comparativa de linguagens de programação sob a ótica linguística revela padrões sintáticos e semânticos que espelham diretamente constructos das línguas naturais. O Portugol, especificamente desenvolvido para falantes do português, exemplifica de maneira explícita esta relação ao empregar termos como "algoritmo", "var", "início" e "fim" que funcionam como equivalentes diretos de categorias gramaticais. Como demonstra Medeiros *et al*. (2020), estudantes que utilizam Portugol mostram maior facilidade na compreensão de conceitos algorítmicos básicos, sugerindo que a proximidade lexical com a língua materna reduz a carga cognitiva no aprendizado de programação. Esta evidência corrobora a hipótese de que a programação opera como um sistema linguístico que pode ser otimizado através do alinhamento com estruturas da língua natural.

A linguagem Ves, analisada por Silva (2019), torna em evidência exemplos concretos de como constructos linguísticos podem ser implementados sintaticamente. Sua estrutura de pattern matching, inspirada em F# e Rust, permite expressões como *match n { 1;2;3 -> print("muito baixo"); x when n % 23 == 0 -> print("23!"); _ -> print("ok"); }* que espelham diretamente construções condicionais das línguas naturais. Essa sintaxe declarativa reduz a distância entre o pensamento algorítmico e sua expressão codificada, funcionando como uma "gramática intermediária" entre língua natural e instrução de máquina. O tratamento de exceções em Ves, com blocos *try-except-finally*, constitui outro exemplo de como estruturas de controle podem ser modeladas a partir de constructos linguísticos de contingência e exceção presentes nas línguas humanas.

Python, por sua vez, ilustra como escolhas de design sintático podem criar pontes cognitivas entre domínios. Sua utilização de estruturas como *list comprehensions [x for x in range(10) if x % 2 == 0]* espelha diretamente construções do português como "a lista de todos os x entre 0 e 9 que são pares". Segundo Van Rossum (1997), esta legibilidade intencional foi um princípio fundamental no design da linguagem, priorizando a clareza sobre a concisão. A sintaxe de dicionários em *Python* {"nome": "João", "idade": 30} também reflete estruturas de pares chave-valor que possuem equivalentes diretos em construções nominais das línguas naturais.

Em linguagens funcionais como Haskell, observa-se a implementação de conceitos linguísticos ainda mais abstratos. O sistema de tipos algébricos permite definições como *data Maybe a = Nothing | Just a* que correspondem diretamente a estruturas de alternância discursiva ("ou é nada, ou é um valor a"). Essa correspondência

entre tipos de dados, união e estruturas de disjunção nas línguas naturais demonstra como categorias lógico-linguísticas fundamentais permeiam tanto a comunicação humana quanto a computacional.

Em Java, o conceito de contrato de métodos – expresso através de interfaces – espelha diretamente os pressupostos conversacionais de Grice (1975), em que falantes compartilham expectativas mútuas sobre o conteúdo e forma das interações comunicativas. A exigência de que classes implementem todos os métodos de uma interface funciona como um equivalente computacional do princípio cooperativo, em que participantes de uma troca comunicativa devem contribuir de maneira adequada ao propósito aceito.

Em linguagens orientadas a objetos como C#, o sistema de propriedades *public string Name { get; set; }* implementa o que na linguística seria analisado como marcadores de acesso e modificação. A distinção entre *getters e setters* corresponde à diferença entre atos ilocucionários assertivos e diretivos na teoria dos atos de fala de Searle (1969). Como observa Silva (2019), em sua análise comparativa, a linguagem Ves explicita essa relação através da sintaxe *prop _.price: float { get; priv set; }* em que os modificadores de acesso funcionam como marcadores pragmáticos que regulam as interações com o objeto.

A programação funcional em linguagens como Elixir torna em evidência exemplos particularmente claros de como conceitos linguísticos podem ser implementados computacionalmente. O *pipe operator* |> que permite encadeamento de funções *1 |> inc() |> double()* espelha diretamente estruturas de tópico-comentário e progressão temática na análise discursiva. Esse paralelo sugere que princípios de coerência e coesão textual, fundamentais para a compreensão de textos em língua natural, possuem equivalentes funcionais na composição de funções em programação.

Em SQL, linguagem dedicada à consulta de bancos de dados, observa-se a implementação de estruturas que correspondem diretamente a atos de fala específicos. O comando *SELECT * FROM users WHERE age > 18* funciona como um equivalente computacional de um ato assertivo que recupera informação, enquanto *INSERT INTO table VALUES (...)* corresponde a um ato declarativo que altera estados de affairs. Essa correspondência entre categorias de comandos SQL e classes de atos ilocucionários demonstra como fundamentos pragmáticos da comunicação humana são formalizados em sistemas computacionais.

A análise de linguagens de domínio específico (DSLs) fornece evidências adicionais da relação entre programação e linguística. A linguagem R, utilizada para análise estatística, implementa construções como *lm(y ~ x1 + x2, data=df)* que espelham diretamente notações matemáticas e linguísticas para relações de dependência. Segundo Manning (2015), esta correspondência entre notação científica, expressão linguística e implementação computacional revela a natureza fundamentalmente linguística do pensamento abstrato, independentemente de seu domínio de aplicação.

O caso do HTML merece menção especial por demonstrar como metáforas linguísticas podem estruturar domínios computacionais inteiros. Sua sintaxe de *tags ... * implementa uma gramática de marcação que espelha estruturas de modificação e qualificação das línguas naturais. A relação entre tags de abertura e fechamento corresponde a estruturas de escopo e dependência sintática analisadas na gramática gerativa de Chomsky (1957).

Em conclusão, a análise detalhada de constructos sintáticos e semânticos em linguagens de programação diversas revela padrões consistentes de correspondência com estruturas linguísticas das línguas naturais. Essas evidências sustentam fortemente a tese de que a programação constitui uma forma de linguística aplicada, em que princípios e estruturas da comunicação humana são formalizados e implementados computacionalmente. A variação entre linguagens de programação reflete, assim, não apenas diferenças técnicas, mas também diferentes abordagens para a modelagem computacional de constructos linguísticos fundamentais.

5 CONSIDERAÇÕES FINAIS

Este artigo buscou demonstrar que a programação de computadores pode ser compreendida como uma forma de linguística aplicada, com base em evidências teóricas, conceituais e práticas. A análise desenvolvida mostrou que a programação compartilha com a linguística preocupações centrais com sintaxe, semântica, pragmática e aquisição da linguagem, além de envolver processos de comunicação e interpretação que são tipicamente humanos. Reposicionar a programação nesse campo mais amplo permite superar visões reducionistas que a limitam à esfera técnica, abrindo espaço para abordagens mais holísticas e humanísticas.

As principais contribuições deste estudo incluem: a articulação de um referencial teórico interdisciplinar que integra ciência da computação e linguística; a análise crítica de linguagens de programação sob a ótica da linguística aplicada; e a proposição de que o ensino de programação pode ser enriquecido por metodologias inspiradas no ensino de línguas. Essas contribuições têm implicações tanto para a pesquisa acadêmica quanto para a prática pedagógica, sugerindo a necessidade de maior diálogo entre áreas tradicionalmente separadas.

Entre as limitações da pesquisa, destaca-se o caráter teórico-conceitual do estudo, que não evidenciou coleta de dados empíricos. Futuras investigações poderiam testar a hipótese proposta por meio de experimentos com aprendizes de programação, observando se estratégias de ensino baseadas em linguística aplicada resultam em melhores outcomes de aprendizagem. Além disso, seria interessante investigar como conceitos linguísticos mais avançados – como análise de *corpus*, linguística cognitiva e teoria da enunciação – poderiam ser aplicados ao estudo de códigos-fonte em larga escala.

Defender que a programação não é tecnologia, mas linguística aplicada, não significa negar sua dimensão técnica, mas antes ampliar seu horizonte de significados. Programar é, antes de tudo, uma atividade simbólica, criativa e comunicativa, que merece ser estudada e praticada com a profundidade e a abrangência que caracterizam as humanidades. Espera-se que este artigo inspire novas reflexões e investigações nessa direção, contribuindo para uma compreensão mais rica e multifacetada da programação no contexto contemporâneo.

REFERÊNCIAS

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compilers**: principles, techniques, and tools. 2. ed. Boston: Pearson Education, 2006.

AUSTIN, J. L. **How to do things with words**. Cambridge: Harvard University Press, 1962.

BENDER, E. M. *et al*. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In: CONFERENCE ON FAIRNESS, ACCOUNTABILITY, AND TRANSPARENCY, 2021. **Proceedings** [...]. New York: ACM, 2021. p. 610–623. DOI: https://doi.org/10.1145/3442188.3445922

BLACKWELL, A. *et al*. The cognitive dimensions of notations. In: COMPANION PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. New York: ACM, 2014. p. 12–21.

CHOMSKY, N. **Syntactic structures**. The Hague: Mouton, 1957.

DIJCK, J. van. **The culture of connectivity**: a critical history of social media. Oxford: Oxford University Press, 2013.

GINZBURG, J. **The interactive stance**: meaning for conversation. Oxford: Oxford University Press, 2012.

GOLDBERG, A. E. **Constructions**: a construction grammar approach to argument structure. Chicago: University of Chicago Press, 1995.

GOLDBERG, Y. **Neural network methods for natural language processing**. Williston: Morgan & Claypool Publishers, 2017. DOI: https://doi.org/10.2200/S00762ED1V01Y201703HLT037.

GRICE, H. P. Logic and Conversation. In: COLE, P.; MORGAN, J. L. (ed.). **Syntax and semantics**: speech acts. New York: Academic Press, 1975. v. 3, p. 41-58.

HALLIDAY, M. A. K. **An introduction to functional grammar**. London: Edward Arnold, 1985.

JURAFSKY, D.; MARTIN, J. H. **Speech and language processing**. 3rd ed. Prentice Hall, 2021.

KNUTH, D. E. **Literate programming**. Stanford: Center for the Study of Language and Information, 1984.

MANNING, C. D. Computational Linguistics and Deep Learning. **Computational Linguistics**, v. 41, n. 4, p. 701-707, 2015. DOI: https://doi.org/10.1162/COLI_a_00239

MARTIN, R. C. **Clean code**: a handbook of agile software craftsmanship. Upper Saddle River: Prentice Hall, 2008.

MEDEIROS, R. P. et al. Um mapeamento sistemático sobre dificuldades de aprendizagem em programação. **Revista Brasileira de Informática na Educação**, v. 28, p. 234–256, 2020. DOI: https://doi.org/10.5753/rbie.2020.28.0.234

MONTAGUE, R. **Formal philosophy**: selected papers of richard montague. New Haven: Yale University Press, 1970.

RAYMOND, E. S. **The Cathedral and the Bazaar**: musings on linux and open source by an accidental revolutionary. Sebastopol: O'Reilly Media, 1999.

ROSCH, E. H. Cognitive Representations of Semantic Categories. **Journal of Experimental Psychology***:* General, v. 104, n. 3, p. 192-233, 1975. DOI: https://doi.org/10.1037/0096-3445.104.3.192

SAUSSURE, F. de. **Cours de linguistique générale**. Paris: Payot, 1916.

SEARLE, J. R. **Speech acts**: an essay in the philosophy of language. Cambridge: Cambridge University Press, 1969.

SILVA, H. R. **Linguagem de Programação Ves**. 2019. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2019.

SOLOWAY, E.; SPOHRER, J. C. **Studying the novice programmer**. New York: Psychology Press, 1989.

SPERBER, D.; WILSON, D. **Relevance**: communication and cognition. Oxford: Blackwell, 1986.

VAN ROSSUM, G. **Comparing Python to other languages**. 1997. Disponível em: https://www.python.org/doc/essays/comparisons/. Acesso em: 15 março 2025.

VASWANI, A. et al. Attention Is All You Need. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 31, 2017. **Proceedings** [...]. Red Hook: Curran Associates, 2017. p. 5998–6008.

WINSKEL, G. **The formal semantics of programming languages**: an introduction. Cambridge: MIT Press, 1993.