Compreensão de Código: Um Problema Antigo em Um Novo Cenário

Valéria Maria Bezerra Cavalcanti Maciel[1,2], **Wilkerson L. Andrade**[2]

[1] Unidade Acadênica de Informática - Instituto Federal da Paraíba (IFPB)
João Pessoa – PB – Brasil

[2]Departamento de Sistemas e Computação - Universidade Federal de Campina Grande (UFCG)
Campina Grande – PB – Brasil

valeria.cavalcanti@ifpb.edu.br, wilkerson@computacao.ufcg.edu.br

Abstract. *This article investigates the level of code comprehension among novice programming students. An empirical study was conducted with 34 participants, involving tasks focused on error correction and problem inference, aiming to assess different dimensions of code understanding. The results show that, although many students are able to correct syntactic and semantic errors, this does not necessarily indicate a genuine understanding of how the code works. Furthermore, most participants struggled to accurately infer the purpose of code snippets presented without any accompanying description. These findings highlight the importance of fostering pedagogical practices that emphasize critical reading and code interpretation—especially in a context where the widespread use of automatic code generation tools has made writing code easier than understanding it.*

Resumo. *Este artigo investiga o grau de compreensão de código por estudantes iniciantes em programação. Por meio de um experimento empírico com 34 participantes, foram aplicadas tarefas de correção de erros e inferência de enunciado, visando avaliar diferentes dimensões da compreensão. Os resultados revelam que, embora muitos estudantes consigam corrigir erros sintáticos e semânticos, isso não garante a compreensão real do funcionamento do código. Além disso, a maioria apresentou dificuldades em identificar corretamente o propósito de trechos de código sem descrição prévia. Esses achados reforçam a importância de promover práticas pedagógicas voltadas à leitura crítica e à interpretação de código, sobretudo considerando o impacto do uso crescente de ferramentas de geração automática de código, contexto em que escrever se tornou mais fácil do que entender.*

1. Introdução

Ao longo da história do ensino de programação, a pesquisa sobre a compreensão de código tem passado por sucessivas transformações em seu foco e abordagem metodológica, adaptando-se às transformações das linguagens e paradigmas. Os primeiros códigos construídos tinham por base um fluxo de execução sequencial, sem grandes desvios ou estruturas complexas. As pesquisas eram baseadas em opiniões, intuições ou experiências pessoais. Esse tipo de abordagem compromete consideravelmente os resultados alcançados, por falta de rigor científico e objetividade.

Em 1979, o trabalho [Sheppard et al. 1979] foi pioneiro ao realizar um estudo empírico na área. Foi concluído que o programador compreende significativamente melhor o código se houver um fluxo lógico claro e direto. Esse estudo destacou a importância de promover diversas experiências de programação com o objetivo de melhorar o desempenho de programadores iniciantes, por exemplo, a depuração de código.

Com o tempo, a área evoluiu e expandiu seu escopo. Investigações qualitativas ganharam espaço, por exemplo, buscando identificar as estratégias utilizadas pelos programadores para analisar os códigos e suas respectivas limitações. Em 2004, o trabalho [Lister et al. 2004] investigou o motivo para o fraco desempenho dos alunos iniciantes em programação. O estudo concluiu que a capacidade de analisar sistematicamente um trecho de código era insatisfatória.

Em busca de uma solução para o problema identificado, os autores em [Lister et al. 2004] sugerem incluir nos currículos a ênfase na habilidade de leitura de código, com rastreamento e raciocínio sobre a intenção do código. Ou seja, esse trabalho destacou a leitura como importante experiência de programação. O mesmo cenário foi encontrado no trabalho [McCracken et al. 2001], que reforçou a necessidade de fortalecer nos estudantes a capacidade de ler sistematicamente um código.

A diversidade de formas para medir a compreensão de código aumentou consideravelmente. Na literatura, são encontrados diversos estudos que investigam as "Tarefas de Compreensão". Essas tarefas são estratégias de ensino que exigem que o estudante demonstre entendimento sobre o funcionamento de um código. Ou seja, ele precisa analisar e explicar seu comportamento e propósito. Algumas dessas tarefas são aprimoradas ao longo do tempo, como "rastreamento de código" e "autoexplicação". Outras são mais intuitivas, como "leitura em voz alta". O foco principal dessas tarefas de compreensão é na leitura e interpretação de códigos.

Em 2018, com o lançamento do BERT (Google) e ChatGPT (OpenAI), começou um novo marco na área de programação. Utilizando os Large Language Models (LLMs), diversos assistentes inteligentes passaram a atuar na geração de código a partir de descrições em linguagem natural. Dessa forma, reduziram consideravelmente a necessidade do programador escrever "manualmente" códigos [Smith et al. 2024].

Essas mudanças, experimentadas por programadores iniciantes e veteranos, estão promovendo profundas transformações no processo de construção de código. A geração de código mais acessível, sem muito esforço intelectual, pode estar contribuindo para aumentar fragilidades cognitivas, principalmente nos programadores iniciantes. Esse potencial problema reforça a importância de adotar estratégias de ensino para ajudar os alunos iniciantes a construir modelos mentais sólidos antes de começarem a escrever código. Entender código não é apenas um requisito técnico, mas uma competência essencial para depurar, modificar e evoluir sistemas.

Em decorrência dessa grande mudança promovida pelas LLMs, é fundamental continuar investigando como os estudantes iniciantes estão desenvolvendo (ou não) a habilidade de compreender código-fonte. Este trabalho se insere nesse contexto e tem como objetivo investigar, por meio de um experimento empírico, o grau de compreensão de código demonstrado por estudantes iniciantes em programação, utilizando tarefas específicas que exigem leitura, análise e interpretação de trechos de código-fonte.

A compreensão de código é uma habilidade essencial para o aprendizado em programação, que vai muito além da simples memorização da sintaxe de uma linguagem [Zavala and Mendoza 2017]. A leitura crítica do código é fundamental para o domínio do problema e consequente construção de sistemas com qualidade e segurança.

2. Fundamentação

Estudos clássicos como o de [Sheppard et al. 1979] já destacavam que a qualidade do software e o desempenho dos programadores estavam diretamente relacionados à clareza do fluxo lógico do código. Ao longo do tempo, a área foi incorporando metodologias mais robustas e os códigos produzidos foram melhorando na legibilidade, consequentemente, compreensão.

Pesquisada há décadas, a compreensão de código é uma habilidade fundamental na área de programação. Estudos como os de [Lister et al. 2004] sugerem que a leitura de código passou a ser valorizada como uma competência fundamental nos cursos introdutórios de programação (CS1). Muitas vezes mais importante que a própria escrita, conforme apontado por [Zavala and Mendoza 2017].

Os autores em [Nelson et al. 2017] também afirmam que experiências práticas de leitura, modificação e depuração favorecem a compreensão, especialmente entre iniciantes. Principalmente em um cenário marcado pela ampla adoção de ferramentas de geração automática de código, como os LLMs.

Em um cenário onde gerar código é mais fácil do que entender, torna-se mais necessário garantir que os estudantes compreendam efetivamente os códigos com os quais interagem, evitando erros decorrentes da interpretação superficial [Leinonen et al. 2023].

Diversos estudos indicam que estudantes iniciantes enfrentam dificuldades significativas ao tentar compreender código-fonte. Essas dificuldades vão além dos aspectos sintáticos da linguagem e envolvem também limitações cognitivas, emocionais e pedagógicas. Trabalhos como os de [Lister et al. 2004] e [McCracken et al. 2001] demonstraram que muitos estudantes não conseguem rastrear ou explicar corretamente o funcionamento de programas simples, o que revela deficiências na construção de modelos mentais.

O estudo de [Hassan et al. 2023] evidencia que estudantes novatos têm dificuldades em acompanhar o código de forma linear, o que pode causar sobrecarga cognitiva. Além disso, muitos estudantes não conseguem explicar nem mesmo seus próprios códigos, conforme relatado por [Lehtinen et al. 2021].

Diante dessas dificuldades, avaliar o grau de compreensão de um estudante é um requisito essencial para determinar sua evolução na área de programação. Na literatura, são encontradas diversas formas de promover essa avaliação.

Estudos como [Eranki and Moudgalya 2016] destacam as tarefas de correção de código em programas fornecidos. Essa é uma tarefa que exige que o estudante compreenda o propósito do código e reconheça inconsistências lógicas ou estruturais.

Em [van der Werf et al. 2022], foram analisadas as explicações oferecidas por estudantes sobre códigos em Python. Esse tipo de tarefa contribui para investigar a profundidade do entendimento de um código.

Como resultado dessas tarefas, é necessário analisar tanto dados quantitativos quanto qualitativos. A taxonomia SOLO (*Structure of the Observed Learning Outcome*) tem sido empregada para avaliar qualitativamente a complexidade das respostas de estudantes em tarefas abertas, por exemplo, explicações sobre o código.

Essa taxonomia classifica as respostas em cinco níveis hierárquicos: pré-estrutural, uniestrutural, multiestrutural, relacional e abstrato estendido. Seu uso permite distinguir estudantes que reconhecem apenas elementos isolados do código daqueles que compreendem sua lógica de funcionamento e conseguem generalizar esse entendimento para novos contextos.

Estudos como os de [Izu et al. 2018], [Mirolo et al. 2020] e [Hassan 2022] aplicaram essa taxonomia em atividades de análise e reversibilidade de código, revelando que muitos estudantes conseguem realizar classificações corretas, mas poucos apresentam justificativas coerentes e completas, o que os coloca em níveis inferiores da escala SOLO.

Com base na revisão sistemática conduzida por [Maciel and Andrade 2025], foram identificados vários trabalhos que abordam a compreensão de código por estudantes iniciantes em programação. Esses estudos apresentam estratégias pedagógicas e uso de ferramentas. Os achados dessa revisão reforçam a importância de integrar diferentes abordagens de ensino e avaliação que contemplem tanto aspectos técnicos quanto cognitivos da aprendizagem da programação.

3. Metodologia

Para investigar o grau de compreensão de código por estudantes iniciantes em programação, foi conduzido um experimento com base na aplicação de uma lista de exercícios desenvolvida especificamente para esse fim.

A proposta permitiu observar a compreensão sob múltiplas perspectivas, associando dados quantitativos e qualitativos. A seguir, são apresentados os participantes, as questões de pesquisa, os detalhes sobre as categorias de exercícios e o desenho do experimento.

3.1. Participantes

O experimento contou com a participação de 48 estudantes matriculados em disciplinas introdutórias de programação, oferecidas nos seguintes cursos de graduação:

- Engenharia de Software
- Engenharia Elétrica
- Engenharia Mecânica
- Sistemas de Telecomunicações
- Sistemas para Internet
- Redes de Computadores

Os estudantes foram convidados por meio de um formulário eletrônico, divulgado pelos professores responsáveis pelas disciplinas introdutórias de programação. A participação foi voluntária, sendo oferecido como incentivo um bônus de pontuação em uma das avaliações da disciplina. Essa estratégia teve como objetivo ampliar a adesão dos estudantes.

Dos 48 estudantes que se inscreveram, 14 foram excluídos da amostra por não atenderem aos critérios estabelecidos. 34 estudantes foram selecionados por estarem em sua primeira experiência formal com a disciplina de programação, mantendo o foco na análise da compreensão de código por estudantes iniciantes.

Esses participantes tiveram contato com os conceitos fundamentais da lógica de programação, estruturas de controle, estruturas de repetição, vetores e matrizes. A escolha desse público se justifica pelo objetivo central do estudo, que é investigar a compreensão de código por programadores em formação, especialmente em seus primeiros momentos de contato com a leitura, interpretação e modificação de código-fonte.

Uma vez apresentado o público-alvo, seguem as questões de pesquisa definidas.

3.2. Questões de Pesquisa

Este estudo foi orientado por três questões de pesquisa. Apresentadas, a seguir:

- Q1: Qual é o grau de compreensão de código-fonte apresentado por estudantes iniciantes em programação?
- Q2: Em que medida os estudantes iniciantes conseguem identificar e corrigir erros sintáticos e semânticos em códigos fornecidos?
- Q3: Os estudantes são capazes de inferir corretamente o enunciado (propósito) de um código (sem uma descrição prévia)?

Com o objetivo de promover a análise fundamentada nas questões de pesquisa, segue a descrição dos exercícios aplicados.

3.3. Categoria de Exercícios

Os exercícios foram organizados em duas categorias principais: Correção e Funcionalidade.

Na categoria **Correção**, os participantes receberam o enunciado e o respectivo código-fonte associado. O objetivo foi localizar e corrigir erros em trechos de código previamente fornecidos, ao final, apresentando o código corrigido. A métrica utilizada foi a quantidade de erros corrigidos corretamente.

Já na categoria **Funcionalidade**, os estudantes receberam apenas o código-fonte e deveriam inferir e descrever qual seria o enunciado correspondente, ou seja, qual problema o código resolvia. Nesse caso, a métrica adotada foi a adequação do enunciado apresentado em relação à funcionalidade real do código.

Essas duas categorias permitiram avaliar diferentes dimensões da compreensão de código, abrangendo tanto aspectos técnicos quanto cognitivos.

Segue o desenho de como o experimento foi realizado.

3.4. Desenho do Experimento

Inicialmente, os estudantes responderam a um questionário com o objetivo de caracterizar o perfil dos participantes e compreender suas experiências e estratégias relacionadas à programação. O questionário investigou aspectos como experiência prévia com programação, estratégias cognitivas utilizadas na leitura de código, motivação para estudar a disciplina e o uso de anotações ou representações gráficas durante a análise de algoritmos.

Além disso, foi solicitado que os estudantes indicassem a ordem preferencial para a resolução das categorias de problemas propostas no experimento, justificando sua escolha. Essas informações forneceram subsídios para contextualizar a análise dos resultados, permitindo observar possíveis correlações entre perfil, estratégias adotadas e desempenho nas atividades.

Em seguida, os participantes receberam uma lista impressa, contendo 04 (quatro) códigos. Igualmente distribuídos nas duas categorias.

Os códigos (C1 e C2) da categoria **Correção** apresentavam um enunciado acompanhado de um trecho de código contendo erros sintáticos e/ou semânticos. Os estudantes deveriam identificar e corrigir esses erros, demonstrando domínio sobre os elementos estruturais da linguagem e a lógica do algoritmo. Os códigos foram impressos coloridos, dessa forma promovendo as vantagens da *syntax highlighting*.

Na categoria **Funcionalidade** os códigos (C3 e C4) forneciam apenas o código-fonte correto, sem qualquer descrição textual. Nesses casos, os estudantes deveriam inferir o enunciado que teria dado origem àquele código, revelando sua capacidade de abstração, interpretação e reconstrução de propósito.

Todas as respostas foram analisadas posteriormente por pesquisadores, com base em critérios definidos para cada categoria: número de erros corrigidos e grau de aderência do enunciado proposto à funcionalidade real do código.

A atividade foi individual, com o tempo para a realização de até 150 (cento e cinquenta) minutos. Os estudantes não puderam realizar consulta a qualquer material.

A seguir, são apresentados os resultados obtidos a partir da aplicação do experimento.

4. Resultados

Os resultados obtidos a partir da análise das respostas dos participantes oferecem uma visão sobre o desempenho dos estudantes em cada uma das categorias de exercício propostas. A seguir, os dados são apresentados separadamente para cada categoria, destacando os padrões observados, os níveis de compreensão demonstrados e as dificuldades mais recorrentes.

4.1. Categoria Correção

Nessa categoria foram apresentados 02 (dois) códigos, o primeiro com 06 (seis) erros e o segundo com 10 (dez) erros. Cada participante foi classificado com base na quantidade total de erros que conseguiu corrigir, de acordo com a Tabela 1.

Nível de Compreensão	Quantidade de Acertos
Baixo	0 a 6
Médio	7 a 11
Alto	12 a 16

Tabela 1. Nível de Compreensão com base no Número de Acertos.

Com base no número de erros corrigidos, em média, 79% foram realizados com sucesso (com mediana de 88%), conforme o Boxplot apresentado na Figura 1. Com base

nesses números e na classificação proposta, 71% dos participantes foram classificados com **ALTO** nível de compreensão.

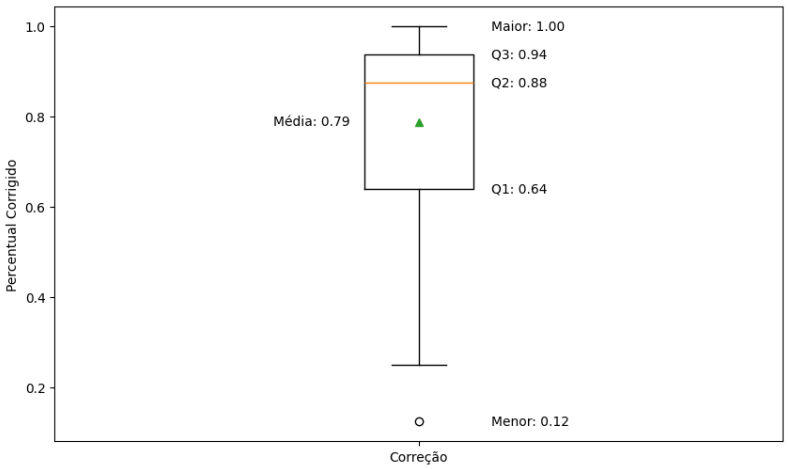

Figura 1. Análise da correção baseada na quantidade de erros corrigidos.

Além do critério quantidade, o código corrigido por cada participante foi avaliado criteriosamente pelos pesquisadores. Cada resposta foi categorizada como "Sim" (indicando compreensão do código) ou "Não" (indicando falta de compreensão). Para compreender melhor essa categorização, segue a análise das principais falhas identificadas nos dois códigos apresentados.

4.1.1. Código C1

Nesse problema, o principal foco a ser avaliado é como os estudantes analisam a estrutura condicional "if". Para tanto, foi utilizado como base o código original observado na Figura 2.

Entre os participantes classificados com ALTO nível de compreensão, 32% apresentavam códigos sintaticamente corretos, mas evidenciaram, nas linhas 9 e 12, que não compreenderam efetivamente o código apresentado.

Esses estudantes demonstram saber usar/corrigir a estrutura condicional "if", no entanto, ao repetir todas as condições, revelam uma compreensão limitada da lógica do código.

4.1.2. Código C2

O foco a ser avaliado nesse problema é como os estudantes analisam a declaração e uso de vetores, conforme código original observado na Figura 3. 18% dos participantes classificados com ALTO nível de compreensão apresentaram incompreensão ao permitir duplicar o tamanho do vetor "notas" (linha 9) e utilizar a média antes de calcular (linha 15).

4.2. Categoria Funcionalidade

Também foram apresentados 02 (dois) códigos (C3 e C4) nessa categoria. Para cada código, os participantes deveriam elaborar um enunciado que poderia ter sido usado pelo

Código C1 - Original

```
n1 = int(input())
n2 = int(input())
n3 = str(input())
n4 = int(input())

if n1 > n2 and n1 > n3 and > n4:
    print(n1)
else
    if n2 > n3 and n2 > n2:
        print(n2)
    else:
        if n3 > n4:
            prinf(n3)
        else:
            print(n1)
```

Código Corrigido

```
1  n1 = int(input())
2  n2 = int(input())
3  n3 = int(input())
4  n4 = int(input())
5
6  if n1 >= n2 and n1 >= n3 and n1 >= n4:
7      print(n1)
8  else:
9      if n2 >= n1 and n2 >= n3 and n2 >= n4:
10         print(n2)
11     else:
12         if n3 >= n1 and n3 >= n2 and n3 >= n4:
13             print(n3)
14         else:
15             print(n4)
```

Figura 2. Principal erro identificado entre os participantes no Código C1

Código C2 - Original

```
nomes = []
notas = [0] * 400

for i in (400):
    nome = input('Informe o nome: )
    nomes.append(nome)
    nota = input('Informe a nota: ')
    notas.append(nota)

for i in range(400)
    if nomes[i] > media:
        qtd =+ 1

media = sum(notas) / 400

print('Média:' media)
print(f'Quantidade de notas acima da média: {qtd}')
```

Código Corrigido

```
1  nomes = []
2  notas = [0] * 400
3  qtd = 0
4
5  for i in (400):
6      nome = input('Informe o nome: ')
7      nomes.append(nome)
8      nota = input('Informe a nota: ')
9      notas.append(nota)
10
11 for i in range(400):
12     if notas[i] > media:
13         qtd =+ 1
14
15 media = sum(notas) / 400.
16
17 print('Média:' media)
18 print(f'Quantidade de notas acima da média: {qtd}')
```

Figura 3. Principal erro identificado entre os participantes no Código C2.

programador que o implementou.

Com o objetivo de não apenas quantificar a correção das respostas, mas também compreender a profundidade da compreensão do código demonstrada por cada participante, foi utilizada a Taxonomia SOLO. Conforme Tabela 2. Dessa forma, foi possível classificar as respostas em níveis hierárquicos, desde uma compreensão superficial até um entendimento profundo e abstrato da lógica e propósito do código.

No primeiro código (C3), 50% dos participantes conseguiram inferir corretamente o enunciado correto, apresentando nível relacional na Taxonomia SOLO. No segundo código (C4), esse número caiu para 11,8%. Na Figura 4, é possível observar a classificação por código.

Nenhum participante atingiu o nível 5 na Taxonomia SOLO.

5. Discussão

Embora 71% dos participantes tenham obtido, com base na quantidade de erros corrigidos, ALTO nível de desempenho na compreensão de código, apenas 30% compreenderam efetivamente os códigos propostos.

Nível	Categoria	Resposta
1	Pré-estrutural	Ausente, irrelevante ou confusa
2	Uniestrutural	Superficial, com foco na entrada e saída dos dados
3	Multiestrutural	Identifica as ações do código, mas não consegue inferir a relação entre elas para buscar o resultado
4	Relacional	Consegue inferir a relação entre as etapas do código para buscar o resultado
5	Abstrato Estendido	Consegue relacionar e reconhecer aplicabilidade em outras ações.

Tabela 2. Nível de Compreensão com base na Taxonomia SOLO.

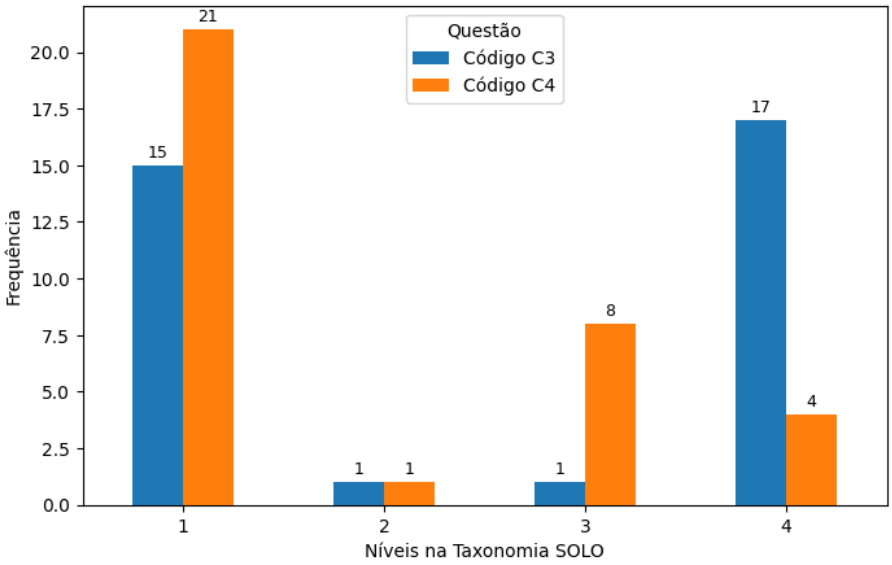

Figura 4. Níveis SOLO dos participantes nos Códigos C3 e C4.

A Figura 5 apresenta a distribuição dos estudantes segundo a compreensão dos Códigos C1 e C2 por Nível de Correção (Baixo, Médio ou Alto). Observa-se que, embora a maioria dos estudantes com desempenho alto tenha demonstrado compreensão dos dois códigos (10 casos na categoria SIM–SIM), uma parcela significativa (14 casos) apresentou desempenho elevado na correção, mas não compreendeu integralmente os códigos, evidenciando dissociação entre correção sintática e compreensão semântica.

Destacam-se os 7 estudantes classificados como NÃO–SIM, que não compreenderam o código C1, mas compreenderam o C2 — um indício de que estratégias superficiais podem ter sido utilizadas na correção. Por outro lado, estudantes com desempenho baixo ou médio tendem a apresentar também baixa compreensão dos códigos, sugerindo maior alinhamento entre os dois critérios.

Esses dados reforçam a importância de avaliar não apenas a capacidade de correção, mas também a compreensão efetiva dos programas, especialmente entre os estudantes que, mesmo obtendo bons resultados em correção, podem não ter entendido efetivamente o código.

Considerando a **Questão de Pesquisa - Q2**, os resultados obtidos sugerem que os

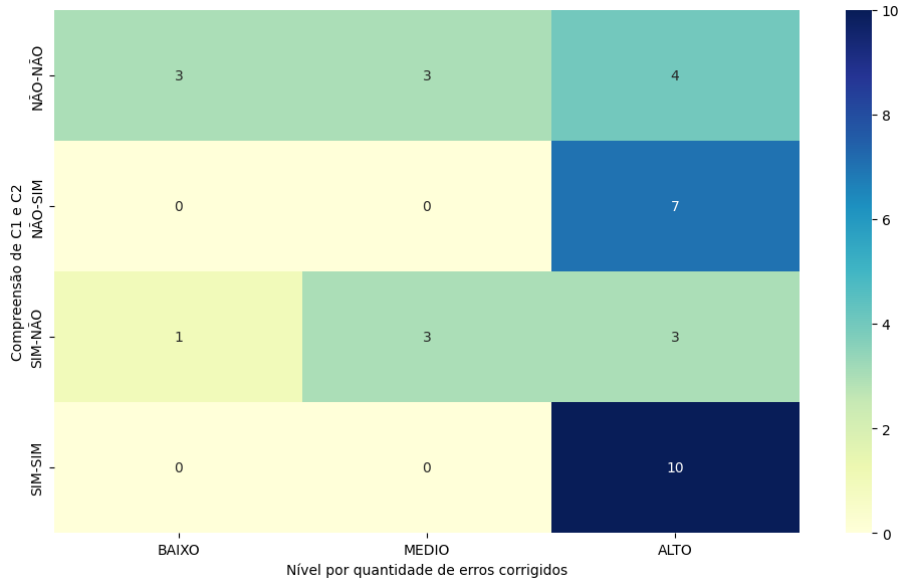

Figura 5. Mapeamento da compreensão dos códigos C1 e C2 por Nível de Correção.

estudantes conseguem corrigir os erros, porém, isso não necessariamente implica afirmar que eles compreendem o código que foi analisado.

A Figura 6 apresenta a distribuição dos estudantes em relação à quantidade de códigos compreendidos na categoria de correção (C1 e C2) e aos níveis da taxonomia SOLO atribuídos às respostas de inferência de enunciado (C3 e C4).

Observa-se que os estudantes que não compreenderam nenhum dos códigos de correção (valor 0) se concentram majoritariamente nos níveis mais baixos da taxonomia, com destaque para o nível 1-1 (pré-estrutural), onde se encontram 7 estudantes.

À medida que aumenta a quantidade de códigos compreendidos na correção, nota-se uma tendência de avanço nos níveis da taxonomia SOLO. Estudantes que compreenderam ambos os códigos de correção são os únicos que atingem o nível 4-4, indicando compreensão relacional dos dois enunciados inferidos.

Esses resultados sugerem uma forte associação entre a compreensão de código na atividade de correção e a construção de enunciados nas tarefas de funcionalidade, evidenciando que a compreensão prévia da estrutura e lógica dos códigos impacta positivamente na qualidade da inferência realizada pelos estudantes.

Considerando a **Questão de Pesquisa Q3**, apenas 11,8% dos participantes conseguiram inferir corretamente o enunciado para os dois problemas propostos, enquanto 21% conseguiram inferir apenas um. A maioria dos erros se concentrou no código C4, cujo código pedia para verificar se um vetor estava ordenado. Ou seja, não é possível afirmar que os estudantes são capazes de inferir corretamente o enunciado (propósito) de um código (sem uma descrição prévia).

Analisando na ótica da **Questão de Pesquisa Q1**, os resultados do experimento indicam que o grau de compreensão de código entre os estudantes iniciantes **ainda é limitado**, especialmente quando analisado de forma qualitativa. Esses achados mostram que o

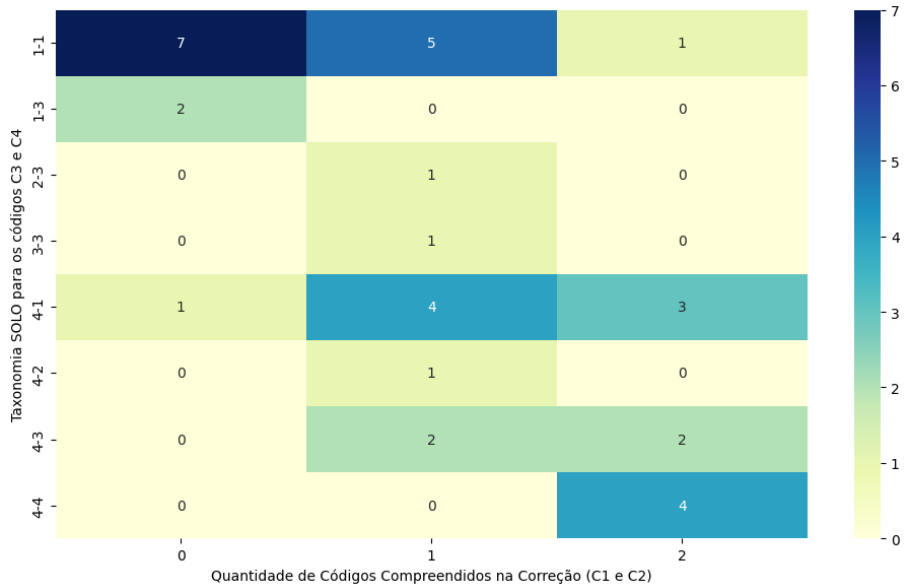

Figura 6. Mapeamento da compreensão dos códigos C1 e C2 por Nível de Correção.

desempenho técnico não necessariamente reflete a compreensão conceitual, o que reforça a importância de utilizar múltiplas abordagens avaliativas para investigar o entendimento real dos estudantes sobre código-fonte.

6. Conclusão

Este estudo buscou investigar o grau de compreensão de código apresentado por estudantes iniciantes em programação. Os resultados revelam que, embora a maioria dos participantes tenha demonstrado bom desempenho na identificação e correção de erros, essa habilidade não garante a compreensão efetiva do funcionamento do código. A análise qualitativa evidenciou que grande parte dos estudantes reproduz correções sem necessariamente compreender a lógica associada ao algoritmo.

Os baixos resultados nos exercícios de inferência do enunciado revelam fragilidade na construção de modelos mentais mais abstratos e relacionais (competências fundamentais para o raciocínio computacional). A correlação entre desempenho nas duas categorias sugere que práticas de leitura crítica e análise estruturada do código devem ser priorizadas no ensino introdutório de programação.

Em um cenário cada vez mais influenciado por ferramentas de geração automática de código, como as LLMs, torna-se ainda mais urgente promover atividades que estimulem a compreensão profunda, a abstração e a interpretação dos programas. Ensinar programação não pode se limitar à produção de código: é preciso formar leitores competentes de algoritmos. Este estudo reforça a necessidade de revisitar estratégias pedagógicas para garantir que, mesmo com o apoio de tecnologias avançadas, o estudante desenvolva autonomia cognitiva para compreender o que o código realmente faz.

Referências

[Eranki and Moudgalya 2016] Eranki, K. L. and Moudgalya, K. M. (2016). Program slicing technique: A novel approach to improve programming skills in novice learners. In *Proceedings of the 17th Annual Conference on Information Technology Education*, SIGITE '16, page 160–165, New York, NY, USA. Association for Computing Machinery.

[Hassan 2022] Hassan, M. (2022). How do we help students "see the forest from the trees?". In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, ICER '22, page 3–4, New York, NY, USA. Association for Computing Machinery.

[Hassan et al. 2023] Hassan, M., Cunningham, K., and Zilles, C. (2023). Evaluating beacons, the role of variables, tracing, and abstract tracing for teaching novices to understand program intent. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 329–343, New York, NY, USA. Association for Computing Machinery.

[Izu et al. 2018] Izu, C., Mirolo, C., and Weerasinghe, A. (2018). Novice programmers' reasoning about reversing conditional statements. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 646–651, New York, NY, USA. Association for Computing Machinery.

[Lehtinen et al. 2021] Lehtinen, T., Lukkarinen, A., and Haaranen, L. (2021). Students struggle to explain their own program code. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, page 206–212, New York, NY, USA. Association for Computing Machinery.

[Leinonen et al. 2023] Leinonen, J., Denny, P., MacNeil, S., Sarsa, S., Bernstein, S., Kim, J., Tran, A., and Hellas, A. (2023). Comparing code explanations created by students and large language models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, page 124–130, New York, NY, USA. Association for Computing Machinery.

[Lister et al. 2004] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '04, page 119–150, New York, NY, USA. Association for Computing Machinery.

[Maciel and Andrade 2025] Maciel, V. M. B. C. and Andrade, W. L. (2025). Code comprehension for novice students: Teaching, assessment, tools, and challenges. In *2025 IEEE Frontiers in Education Conference (FIE) (FIE 2025)*, page 9, Nashville, USA.

[McCracken et al. 2001] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '01, page 125–180, New York, NY, USA. Association for Computing Machinery.

[Mirolo et al. 2020] Mirolo, C., Izu, C., and Scapin, E. (2020). High-school students' mastery of basic flow-control constructs through the lens of reversibility. In *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, WiPSCE '20, New York, NY, USA. Association for Computing Machinery.

[Nelson et al. 2017] Nelson, G. L., Xie, B., and Ko, A. J. (2017). Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, page 2–11, New York, NY, USA. Association for Computing Machinery.

[Sheppard et al. 1979] Sheppard, Curtis, Milliman, and Love (1979). Modern coding practices and programmer performance. *Computer*, 12(12):41–49.

[Smith et al. 2024] Smith, D. H., Denny, P., and Fowler, M. (2024). Prompting for comprehension: Exploring the intersection of explain in plain english questions and prompt writing. In *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, L@S '24, page 39–50, New York, NY, USA. Association for Computing Machinery.

[van der Werf et al. 2022] van der Werf, V., Aivaloglou, E., Hermans, F., and Specht, M. (2022). What does this python code do? an exploratory analysis of novice students' code explanations. In *Proceedings of the 10th Computer Science Education Research Conference*, CSERC '21, page 94–107, New York, NY, USA. Association for Computing Machinery.

[Zavala and Mendoza 2017] Zavala, L. and Mendoza, B. (2017). Precursor skills to writing code. *J. Comput. Sci. Coll.*, 32(3):149–156.