Quebrando o Silêncio Pedagógico: O Modelo WOKDEX para Feedback Formativo em Juízes Online

Alessio Miranda Junior[1,2] **, Vinicius Fernandes dos Santos**[2]

[1]Departamento de Computação (DECOM-TM) – CEFET-MG – Timóteo – MG – Brasil

[2]Departamento de Ciência da Computação (DCC) – UFMG – BH – MG – Brasil

alessio@cefetmg.br, viniciussantos@dcc.ufmg.br

Abstract. *Online programming judges operate under "pedagogical silence": their binary verdicts* ***(e.g., WA, TLE)*** *neither guide the student nor distinguish correct logic from accidental passes. This paper presents WOKDEX, a metadata schema that organizes test cases with explicit formative intent. The model proposes a test typology separating examples (*`SAMPLE`*), functional tests (*`FUNCTIONAL`*), misconception detection (*`MISCONCEPTION`*), and complexity analysis (*`PERFORMANCE`*). The* ***Hint Modulation Heuristic (HMH)*** *modulates feedback based on error consistency. Applied to 45 curated exercises, WOKDEX demonstrates how to embed pedagogical intent directly into the exercise artifact, addressing a critical gap in current packaging standards.*
Keywords: *Pedagogical Metadata, Online Judges, Formative Feedback, Programming Education, Test Typology.*

Resumo. *Juízes online de programação operam sob "silêncio pedagógico": seus vereditos binários (ex: WA, TLE) não orientam o estudante nem distinguem lógicas corretas de aprovações acidentais. Este artigo apresenta o WOKDEX, um schema de metadados que organiza casos de teste com intenção formativa explícita. O modelo propõe uma tipologia de testes que separa exemplos (*`SAMPLE`*), testes funcionais (*`FUNCTIONAL`*), detecção de misconceptions (*`MISCONCEPTION`*) e análise de complexidade (*`PERFORMANCE`*). A* ***Heurística de Modulação de Dica (HMD)*** *modula o feedback com base na consistência dos erros. Aplicado a 45 exercícios curados, o WOKDEX demonstra como embutir a intenção pedagógica diretamente no artefato do exercício, preenchendo uma lacuna crítica dos formatos de empacotamento atuais.*
Palavras-chave: *Metadados Pedagógicos, Juízes Online, Feedback Formativo, Ensino de Programação, Tipologia de Testes.*

1. Introdução

O ensino de programação enfrenta uma contradição estrutural: a aprendizagem efetiva exige que o estudante escreva código, erre, compreenda o erro e tente novamente [Robins et al. 2003] — um ciclo que só produz crescimento se mediado por feedback formativo. Contudo, em disciplinas numerosas, essa mediação individual é operacionalmente inviável: nenhum docente dispõe de tempo para analisar cada raciocínio. Mesmo com professores assistentes, o modelo permanece 1-para-1 e não escala. Além da inviabilidade, a correção manual não produz dados sistematizados: padrões de erro ficam dispersos, impedindo o diagnóstico coletivo e a melhoria dos exercícios [Ihantola et al. 2010].

A automação da avaliação surge, portanto, como condição necessária para viabilizar o feedback formativo em escala.

A literatura organiza as soluções de avaliação automática em dois paradigmas [Wasik et al. 2018, Paiva et al. 2022]. Os **juízes online** (ex: Boca, DOMJudge, Codeforces, Kattis, LeetCode, Beecrowd) [de Campos and Ferreira 2004, Mariani et al. 2005, Mirzayanov et al. 2020, Kattis 2016, LeetCode 2015, Beecrowd 2024] originaram-se na programação competitiva: avaliam por comparação de I/O e emitem vereditos binários (*Accepted*/*Wrong Answer*). Sua força está na escalabilidade; sua fraqueza, na opacidade pedagógica. Já os **autograders** acadêmicos incorporam testes unitários e feedback rubricado em plataformas fechadas. Sua força é a riqueza diagnóstica; sua fraqueza, o aprisionamento (*platform-lock*) — a inteligência reside no sistema, perdendo-se ao migrar exercícios. Revisões confirmam que a vasta maioria das ferramentas ainda se restringe à avaliação somativa [Messer et al. 2024]. Embora a teoria exija **avaliação formativa** para o desenvolvimento contínuo [Black and Wiliam 1998], ambos os paradigmas reduzem a avaliação de código (uma tarefa discursiva) ao equivalente de checar se uma redação termina com ponto final.

Essa limitação traz consequências práticas: a literatura documenta taxas de reprovação superiores a 30% em turmas introdutórias [Watson and Li 2014], evidenciando a insuficiência do veredito binário. Em nossa análise retrospectiva de 2.654 submissões legadas (tratadas como dados secundários desidentificados para garantir a conformidade ética), a maioria dos vereditos opacos (como *Wrong Answer* e *Time Limit Exceeded*) não resultou em correção imediata. Forma-se um ciclo vicioso: sem orientação, o aluno recorre a tentativas aleatórias (*guess-checking*); o professor, sem tempo, aceita a métrica superficial; e o déficit cognitivo só é descoberto na prova presencial, quando já é tarde para intervenções formativas.

O problema mais grave do juiz online tradicional é o desperdício de dados formativos perante falhas conceituais sistemáticas (*misconceptions*): mesmo quando a submissão recebe um veredito opaco de *Wrong Answer* (WA), muitas vezes seria possível realizar uma inferência abdutiva de alta probabilidade para identificar exatamente o modelo mental incorreto do estudante. Por exemplo, um aluno que declara variáveis numéricas como `int` em vez de `double` obterá sucesso onde a divisão for exata, mas falhará sistematicamente em casos fracionários. O veredito binário de WA apenas pune a falha sem investigar o padrão numérico da saída incorreta. Nenhum formato de empacotamento vigente prevê mecanismos de inversão semântica para capitalizar sobre a previsibilidade desse WA e transformá-lo em um diagnóstico preciso.

A segunda limitação estrutural manifesta-se no veredito *Time Limit Exceeded* (TLE): ele não distingue se a solução é logicamente incorreta ou se é correta mas com complexidade inadequada. Para o estudante, ambos os casos são indistinguíveis, gerando frustração e depuração mal direcionada. A separação entre **corretude lógica** e **eficiência algorítmica** é pedagogicamente essencial — especialmente em CS2 e CS3, onde o domínio assintótico é competência curricular explícita [ACM/IEEE 2013, ACM/IEEE 2020] — mas inexiste nos formatos de avaliação atuais.

O diagnóstico comum a ambas as limitações é a **ausência de metadados pedagógicos** nos formatos de empacotamento de exercícios [ICPC 2024]. Formatos como ICPC,

Kattis e Polygon tratam todos os casos de teste como equivalentes: não modelam a intenção pedagógica de cada cenário, não distinguem testes que detectam *misconceptions* de testes que validam comportamento funcional, e não separam cenários de performance dos demais. Os sistemas que avançam em direção ao feedback formativo — como CodeRunner [Lobb and Harlow 2016], Web-CAT [Edwards and Pérez-Quiñones 2008] e Feeper [Alves and Jaques 2014] — resolvem parcialmente o problema, mas mantêm a inteligência pedagógica na *plataforma*, não no *exercício*: ao migrar o artefato entre sistemas, toda a camada diagnóstica se perde.

Este artigo propõe o **WOKDEX**, um schema de metadados que emite a intenção pedagógica diretamente no artefato do exercício, conferindo-lhe a portabilidade dos juízes online e a riqueza diagnóstica dos autograders. O schema organiza os casos de teste em cenários tipados segundo sua função pedagógica — distinguindo exemplos visíveis, testes funcionais, detectores de *misconceptions* e avaliadores de complexidade — e complementa essa tipologia com uma Heurística de Modulação de Dica (HMD) que modula a emissão de feedback com base na consistência dos erros observados. Por ser agnóstico de plataforma e validado por JSON Schema, o WOKDEX é complementar aos padrões existentes: um exercício empacotado com WOKDEX pode ser distribuído via LTI e rastreado via xAPI sem conflito. As contribuições deste artigo são sintetizadas na Tabela 1.

Tabela 1. Contribuições do artigo

ID	Contribuição
C1	Proposta do schema WOKDEX, um modelo de metadados estruturado (YAML validado por JSON Schema) que formaliza atributos pedagógicos para exercícios de programação.
C2	Formalização de uma tipologia de testes que categoriza cenários segundo sua intenção pedagógica: `SAMPLE`, `FUNCTIONAL`, `MISCONCEPTION` (detecção de *misconceptions*) e `PERFORMANCE` (complexidade algorítmica).
C3	Proposição e catalogação de um *corpus* de referência composto por 45 exercícios reais (230 cenários), adaptados sob a taxonomia proposta para os níveis CS1, CS2 e CS3 [ACM/IEEE 2013, ACM/IEEE 2020], com disponibilização pública em formato aberto.
C4	Proposta da Heurística de Modulação de Dica (HMD), que modula a confiabilidade do feedback emitido com base na taxa de falha intra-cenário.

2. Fundamentação Teórica

2.1. Misconceptions e Código Acidentalmente Correto

Misconceptions em programação são entendimentos incorretos mas persistentes sobre o comportamento de construções da linguagem. Elas podem ser classificadas em três categorias [Silva 2024]:

- **MC[1]**: Misconceptions que causam erro, detectáveis por qualquer juiz;
- **MC[2]**: Misconceptions que causam erro parcial, detectáveis com testes abrangentes;
- **MC[3]**: Misconceptions in Correct Code, *invisíveis* ao modelo binário.

É a categoria MC^3 que motiva a criação do tipo de teste `MISCONCEPTION` — cenários projetados deliberadamente para apontar erros lógicos em códigos que, de outra forma, seriam aprovados acidentalmente por testes convencionais.

2.2. Feedback Formativo e Zona de Desenvolvimento Proximal

A Teoria da Zona de Desenvolvimento Proximal (ZPD) [Vygotsky 1978] postula que a aprendizagem ocorre mais efetivamente quando o aprendiz recebe suporte adequado na fronteira entre o que já domina e o que ainda não consegue. Em revisão seminal sobre ensino de programação [Robins et al. 2003], demonstra-se que novatos dependem de prática guiada e feedback iterativo para construir modelos mentais corretos, diferenciando-se de programadores experientes que já possuem repertório de padrões. No contexto de juízes online, o feedback binário opera *fora* da ZPD: não fornece andaime (*scaffolding*) suficiente para que o estudante avance autonomamente.

O feedback pode ser classificado em três níveis progressivos de elaboração: verificação, elaboração e andaime [Shute 2008]. Modelos de feedback (corretivo, epistêmico, sugestivo) em ambientes virtuais de aprendizagem evidenciam que a eficácia do suporte depende diretamente de sua adequação ao nível cognitivo do aprendiz [Costa et al. 2016]. Consequentemente, a avaliação formativa figura como um dos fatores de maior impacto na aprendizagem [Black and Wiliam 1998]. A Tabela 2 contrasta as capacidades de feedback entre juízes tradicionais e o WOKDEX.

Tabela 2. Níveis de feedback (Shute, 2008) aplicados a juízes online

Nível de Feedback	**Juiz Tradicional**	**WOKDEX**
Verificação (certo/errado)	✓ AC/WA binário	✓ AC/WA enriquecido com `testType` e `level` (A–D), identificando *em qual cenário* o erro ocorreu
Elaboração (por que errou)	✗ Silêncio total	✓ `helpTip` contextualizado por cenário, com dica específica para a misconception detectada
Diagnóstico (qual o erro)	✗ Não distingue tipos de erro	✓ `MISCONCEPTION` isola a misconception; consistência multi-caso confirma o padrão
Andaime (como corrigir)	✗ Silêncio total	✓ Editorial do exercício + mapeamento de `skill gap` para orientação curricular

No caso de MC[3], o problema é ainda mais grave: o feedback é *incorretamente positivo*, reforçando entendimentos equivocados.

3. Trabalhos Relacionados

A análise estática já foi proposta para prover feedback em CS1 [Edwards 2004], mas com foco em estilo e boas práticas, não em *misconceptions* algorítmicas. O mapeamento sistemático de mais de 100 ferramentas de feedback automático identifica que a grande maioria não distingue a natureza dos tipos de erro [Keuning et al. 2018]. O CodeRunner integra testes ao Moodle mas não contempla o conceito de falso positivo. Como base para superar essa limitação, os padrões de erro de programadores novatos encontram-se extensivamente documentados, fornecendo lastro empírico para a taxonomia de *misconceptions* [Robins 2019].

Grandes Modelos de Linguagem (LLMs) em arquiteturas de *Retrieval-Augmented Generation* (RAG) têm sido explorados como tutores. Neste cenário, o WOKDEX não concorre com a IA, mas atua como seu *grounding* semântico ideal: ao injetar metadados pedagógicos no contexto, o schema mitiga alucinações e direciona o foco do LLM estritamente para a *misconception* mapeada, guiando o estudante sem violar sua Zona de Desenvolvimento Proximal.

3.1. Padrões de Interoperabilidade Educacional

O intercâmbio educacional baseia-se em padrões consolidados (SCORM [ADL Initiative 2004], IMS QTI [IMS Global Learning Consortium 2013], xAPI [ADL Initiative 2015], LTI [IMS Global Learning Consortium 2019]) que, embora resolvam a distribuição em LMS, não abordam a semântica interna do exercício de programação. Iniciativas como o PExIL [Queirós and Leal 2012] pavimentaram o caminho para a interoperabilidade estrutural desses artefatos via XML. Contudo, conforme ilustra a Tabela 3, o WOKDEX avança ao propor uma camada ortogonal de interoperabilidade pedagógica.

Tabela 3. Posicionamento do WOKDEX no ecossistema de padrões educacionais

Padrão	Camada	Problema	Limitação para exercícios de programação
SCORM	Conteúdo	Portabilidade de módulos	Orientado a conteúdo estático; sem semântica de exercícios ou casos de teste
IMS QTI	Questão	Intercâmbio de avaliações	Projetado para questões objetivas; sem suporte a execução dinâmica ou intenção de cenários
xAPI	Evento	Rastreamento de experiências	Registra *o que ocorreu*; não define a estrutura nem os metadados pedagógicos do exercício
IMS LTI	Protocolo	Integração com LMS/ferramenta	Resolve a conexão entre sistemas; agnóstico quanto ao formato interno do artefato
ICPC Package	Artefato	Portabilidade técnica	Estrutura técnica de I/O; sem intenção pedagógica dos cenários
ProFormA	Artefato	Configuração de *autograders*	XML rico em configurações; sem suporte a tipificação de *misconceptions*
WOKDEX	**Artefato**	**Portabilidade pedagógica**	**—** (preenche a lacuna)

No âmbito dos formatos de empacotamento, o padrão ICPC [ICPC 2024] e suas variantes (Kattis, Polygon) definem a estrutura de arquivos para distribuição de exercícios competitivos. Historicamente, a programação competitiva desenha testes para "quebrar" soluções por performance e casos de borda (avaliação somativa rigorosa), tratando todos os testes como equivalentes. No contexto acadêmico, o formato europeu ProFormA [Striewe and Balz 2015] avança ao usar XML para empacotar exercícios destinados a *autograders*, possuindo ricas anotações de configuração. Entretanto, o ProFormA não tipifica a intenção diagnóstica dos testes baseada em *misconceptions* (MC^3). Em contraste, o WOKDEX preenche essa lacuna com cenários tipados (como o `MISCONCEPTION`) projetados explicitamente para diagnosticar modelos mentais errados (avaliação formativa). Paralelamente, o framework ACM/IEEE CS2013 [ACM/IEEE 2013] e o CC2020 [ACM/IEEE 2020] fornecem taxonomias de competências, porém sem mecanismo operacional para vinculá-las a cenários de teste específicos.

A teoria de MC^3 [Silva 2024] é a mais próxima conceitualmente, mas não propõe uma formalização operacional (campo de metadados, integração com schema, feedback automatizado). O WOKDEX pode ser visto como a **implementação técnica** que operacionaliza o conceito teórico de MC^3, adicionando ainda a dimensão de `PERFORMANCE` e a heurística de modulação HMD.

4. A Arquitetura do Schema WOKDEX e sua Tipologia

O WOKDEX é um modelo de metadados composto por uma **estrutura padronizada de diretórios** e um **arquivo de metadados YAML** validado por um JSON Schema. O diretório do exercício contém os arquivos de entrada e saída esperada organizados por cenário, enquanto o YAML descreve os atributos pedagógicos de cada cenário. O schema organiza os atributos em três camadas: **Descritiva** (informações gerais), **Pedagógica** (habilidades e dicas) e **Avaliativa** (cenários de teste tipados).

4.1. Camada Descritiva

Contém campos como `title`, `level` com valor CS1, CS2 ou CS3, conforme os referenciais curriculares da ACM [ACM/IEEE 2013, ACM/IEEE 2020], `difficulty` e `tags`. Esses campos são análogos aos encontrados em formatos existentes como Kattis e Polygon.

4.2. Camada Pedagógica: Habilidades e Dicas

Cada exercício é anotado com um array de `skills` que mapeia as competências cognitivas necessárias para sua resolução. O schema sugere, como ponto de partida, o uso de taxonomias alinhadas aos referenciais curriculares da ACM, como o CS2013 [ACM/IEEE 2013] e o CC2020 [ACM/IEEE 2020], que organizam competências em áreas de conhecimento hierárquicas (ex: `data-types > casting`, `sorting > stability`). Essa escolha não é prescritiva: o campo `skills` aceita qualquer vocabulário controlado, permitindo que cada instituição adote ou estenda a taxonomia conforme suas necessidades curriculares. Adicionalmente, cada cenário de teste possui um campo `helpTip` que contém uma dica formativa contextualizada, exibida ao estudante em caso de falha.

4.3. Camada Avaliativa: Tipologia Estendida de Testes

A principal inovação do schema é o campo `testType`, que classifica cada cenário de teste segundo sua intenção pedagógica. Cada cenário recebe um `level` (A–D) que indica sua dificuldade: `D` (exemplos do enunciado), `C` (funcionais complementares), `B` (casos especiais/bordas) e `A` (testes exigentes e performance). Assim como a taxonomia de `skills`, esta escala A–D é uma sugestão de implementação (não estática ou restritiva), permitindo que instituições a adaptem para outros níveis de complexidade. O Listing 1 ilustra dois cenários reais do exercício “Divisão”:

```yaml
testScenarios:
  - id: d-sample
    name: Exemplos
    level: D
    testType: SAMPLE
    helpTip: "Verifique os testes basicos do enunciado."
    skills:
      - { skill: io, points: 1 }
      - { skill: variables, points: 1 }
  - id: c-falso-positivo-inteiros
    name: Inteiros - Falso Positivo
    level: C
    testType: MISCONCEPTION
    targetLanguages: ["C", "C++", "Java"]
    helpTip: "Erro na declaracao de tipo de variavel."
    skills:
      - { skill: variables, points: 1 }
```

Listing 1. Exemplo de cenários no metadata.yaml

O campo `id` de cada cenário corresponde ao nome de um diretório dentro da estrutura do exercício. Esse diretório contém os pares de arquivos de entrada (`.in`) e saída esperada (`.out`) que compõem os casos de teste do cenário. Dessa forma, os metadados pedagógicos (YAML) e os dados de teste (arquivos) coexistem na mesma estrutura de diretórios, sem acoplamento. Os tipos de teste disponíveis são apresentados na Tabela 4.

Tabela 4. Tipologia de testes do WOKDEX

Tipo	Visível?	Descrição
`SAMPLE`	Sim	Testes correspondentes aos exemplos do enunciado, utilizáveis para depuração local antes da submissão
`FUNCTIONAL`	Não	Teste funcional secreto que valida um comportamento específico não revelado ao estudante
`MISCONCEPTION`	Não	Teste projetado para invalidar soluções acidentalmente corretas (detecção de *misconceptions*)
`PERFORMANCE`	Não	Teste que avalia complexidade algorítmica (tempo, memória, saída), separando-a da corretude lógica

Essa tipologia, combinada com a hierarquia de níveis, permite que o motor de avaliação **diagnostique a natureza do erro**, não apenas sua existência, e apresente o feedback na ordem pedagógica adequada — do mais simples ao mais complexo.

4.4. Heurística de Modulação de Dica (HMD)

A emissão de um `helpTip` nem sempre é pedagogicamente adequada: se o estudante erra apenas 1 de 10 casos de um cenário, o erro pode ser pontual e a dica generalista pode ser enganosa. Para modular essa decisão, propomos a **Heurística de Modulação de Dica** (HMD), calculada a partir da proporção de falhas sobre o total de casos de teste do cenário. O motor opera com três faixas: **alta confiança** (todos os casos falharam) emite a dica diretamente; **confiança moderada** (metade ou mais) emite a dica qualificada com "*Possível causa:*"; **baixa confiança** delega a uma dica genérica ou ao módulo de IA. Reconhecemos que este é um modelo linear rudimentar e trata-se de uma proposta empírica com limiares projetados para calibração iterativa com dados reais de uso [Hake 1998] — não de uma métrica estatística rigorosa.

A Figura 1 ilustra o fluxo de avaliação *fail-fast* proposto, mostrando como os quatro tipos de teste são processados sequencialmente para produzir diagnósticos diferenciados.

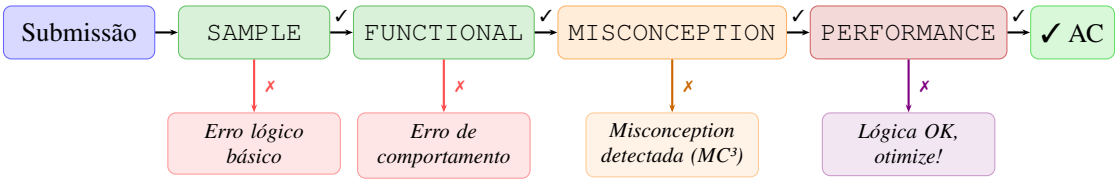

Figura 1. Fluxo de avaliação *fail-fast* com diagnóstico diferenciado por tipo de teste. (Nota: a transição ✓ indica que a submissão sobreviveu à etapa; no cenário `MISCONCEPTION`, significa que o código *não* ativou o gabarito de erro).

4.5. MISCONCEPTION: Definição Formal

Um docente experiente conhece, por vivência acumulada, os erros recorrentes dos estudantes — falhas conceituais que um juiz tradicional reportaria como um genérico *Wrong*

Answer. Esse conhecimento tácito pode ser **codificado como teste**: ao escolher deliberadamente entradas que exponham uma *misconception* específica, o autor transforma sua intuição pedagógica em um artefato verificável e reutilizável. Além disso, o histórico de submissões permite identificar empiricamente padrões de erro não antecipados, retroalimentando o exercício com novos cenários `MISCONCEPTION` e promovendo uma evolução contínua do modelo. É essa a premissa deste tipo estendido.

Definição. O cenário `MISCONCEPTION` opera por **inversão semântica** no motor de avaliação. Seus diretórios mantêm a estrutura legada de juízes online, mas os arquivos de saída (`.out`) não armazenam o gabarito correto; eles armazenam a **Saída Incorreta Previsível** da *misconception*. Se a solução do estudante obtiver um *match* exato (um falso *Accepted*) contra esses arquivos manipulados, o WOKDEX interpreta o evento como a confirmação diagnóstica do erro cognitivo, mantendo o *schema* YAML limpo. Propomos uma taxonomia de três subtipos (Tabela 5).

Tabela 5. Taxonomia dos subtipos de MISCONCEPTION

Subtipo	Misconception	Exemplo	Nível
`TYPE`	Tipo de dado inadequado	`int` em vez de `double` para divisão; saída `5` em vez de `5.00`	CS1
`FORMAT`	Formatação de saída incorreta	`println` em vez de `printf("%.2f")`; separador vírgula em vez de ponto	CS1
`STRATEGY`	Estratégia algorítmica insuficiente	Estratégia Gulosa em cenário de Programação Dinâmica; BFS em grafo ponderado em vez de Dijkstra	CS2–CS3

4.6. PERFORMANCE: Definição Formal

Definição. Cenários `PERFORMANCE` validam a eficiência do código através de entradas dimensionadas para esgotar limites de recursos — tempo (*Time Limit Exceeded*, TLE), memória (MLE) ou saída (OLE) — em algoritmos subótimos. Eles **pressupõem a prévia validação lógica** (`SAMPLE` e `FUNCTIONAL`).

Graças a essa separação e à arquitetura *fail-fast*, a ambiguidade do TLE é resolvida: um *loop* infinito falhará precocemente nos testes funcionais, garantindo que falhas isoladas em `PERFORMANCE` denotem ineficiência assintótica. Isso permite ao motor trocar vereditos genéricos por feedbacks precisos, como: "*Lógica correta, mas a complexidade* $O(N^2)$ *deve ser otimizada para* $O(N \log N)$".

4.7. Dependência da Linguagem e Semântica

Embora o schema WOKDEX seja tecnologicamente agnóstico, a instanciação empírica dos testes e as dicas geradas para cenários `MISCONCEPTION` são inerentemente dependentes da semântica da linguagem alvo. Por exemplo, a *misconception* da divisão com tipos inteiros (abordada na Seção 5) é crítica em C/C++ e Java, mas não ocorre em Python 3, onde o operador `/` sempre retorna um `float`. Para mitigar o paradoxo do acoplamento e garantir maturidade arquitetural, o schema WOKDEX implementa a propriedade restritiva `targetLanguages` (ver Listing 1). O autor pode declarar `targetLanguages: ["C", "C++", "Java"]` no escopo de um cenário `MISCONCEPTION`, instruindo o juiz online a ignorar aquele diagnóstico caso o aluno submeta uma solução em Python. Isso previne *feedbacks* espúrios em linguagens nas quais a *misconception* alvo sequer se manifesta.

5. Exemplos em Três Níveis Curriculares

5.1. CS1: Divisão com Tipo Inteiro (Exemplo Detalhado)

O exercício "Divisão" solicita leitura de dois inteiros e exibição do resultado com duas casas decimais. A *misconception* típica em CS1 é declarar variáveis como `int` em vez de `double`: ao escrever `(double)(a/b)`, o estudante aplica o cast *após* a divisão inteira, de modo que `(double)(10/2)` produz `5.00` — aprovado pelo juiz — mas `(double)(19/6)` produz `3.00` em vez de `3.17`.

Comportamento:

- **Teste SAMPLE** (10, 2): $10/2 = 5 \rightarrow$ `5.00` ✓ *(falso verde)*
- **MISCONCEPTION** (19, 6): $19/6 = 3$ (inteiro) $\rightarrow$ `3.00` ✗ (esperado: `3.17`)
- **MISCONCEPTION** (1, 3): $1/3 = 0$ (inteiro) $\rightarrow$ `0.00` ✗ (esperado: `0.33`)

O aspecto crucial para evitar falsos diagnósticos (ex: o estudante imprimir um erro aleatório) reside nesta arquitetura invertida. O juiz exige um *match* estrito contra os arquivos `.out` do cenário `MISCONCEPTION`, preenchidos deliberadamente com o valor truncado. Quando a submissão coincide perfeitamente com esses gabaritos subótimos em um conjunto de N testes distintos dentro do mesmo diretório, a probabilidade estatística de um erro sintático gerar essa exata coincidência matemática sucessivas vezes é nula. É essa *triangulação comportamental* que blinda o diagnóstico.

Feedback WOKDEX: "*Erro na declaração de tipo de variável. Suas variáveis são inteiras, mas a divisão pode gerar resultado decimal. Verifique se está usando double/float.*"

5.2. CS2 e CS3: Síntese dos Exemplos

A Tabela 6 sintetiza os exemplos de CS2 e CS3, seguindo o mesmo padrão diagnóstico demonstrado em CS1.

Tabela 6. Exemplos de MISCONCEPTION e PERFORMANCE em CS2 e CS3

Nível	Tipo	Misconception	Padrão Diagnóstico
CS2	MC	BFS em grafo ponderado	SAMPLE (pesos iguais): ✓ MC (pesos 1,5,2): *Match* com a minimização de arestas em vez do custo ✗
CS3	MC	Estratégia Gulosa (*Greedy*) em problema de Prog. Dinâmica	SAMPLE (escolha gulosa acerta o ótimo): ✓ MC (ex: troco 6 c/ moedas 1,3,4): *Match* com o subótimo previsível (3) em vez do ótimo (2) ✗
CS3	PERF	Dijkstra $O(V^2)$ sem heap	SAMPLE ($N \leq 100$): ✓ PERF ($N = 10^5$): $O(V^2)$ excede tempo limite ✗

MC = `MISCONCEPTION`; PERF = `PERFORMANCE`

6. Proposição e Instanciação do Corpus

Para demonstrar a expressividade e viabilidade prática do schema WOKDEX, construímos e disponibilizamos um *corpus* de referência composto por 45 exercícios curados (230 cenários de teste, com um mínimo de 4 por problema). Estes foram distribuídos da seguinte forma: [XX] exercícios em CS1, [YY] em CS2 e [ZZ] em CS3, selecionados com base na sua capacidade de ilustrar as falhas cognitivas mais representativas da literatura.

Ressalta-se que não se trata de enunciados inéditos: buscou-se reclassificar e anotar com os metadados do WOKDEX problemas pré-existentes de bases da Olimpíada Brasileira de Informática (OBI) e da Maratona de Programação (ICPC). Todo este *corpus* encontra-se integralmente disponibilizado em repositório aberto[1]. É importante destacar que os cenários `MISCONCEPTION` não foram criados de forma arbitrária; eles foram projetados especificamente para capturar os padrões de erro empíricos observados na amostra histórica detalhada na validação *in-silico* (Seção 7.4), utilizando uma ferramenta própria de autoria para viabilizar a anotação estruturada.

Durante o processo de modelagem e instanciação deste *corpus*, observamos que a aplicação da tipologia estendida atua como um andaime pedagógico que se adapta naturalmente ao nível curricular. Ela cria uma **régua de sensibilidade e gradatividade na correção**, conforme sintetizado na Tabela 7:

Tabela 7. Papel dos tipos estendidos por nível curricular

Nível	Foco dos cenários estendidos	Papel do `PERFORMANCE`
CS1	`MISCONCEPTION` detecta *misconceptions* clássicas (casting, formatação, escopo) que passariam invisíveis ao modelo binário	Pouco relevante: tipicamente exercícios de complexidade $O(1)$ ou $O(N)$
CS2	Separa erro lógico de ineficiência estrutural; valida escolha adequada de algoritmo	Relevante: ordenação e busca exigem conhecimento de complexidade assintótica
CS3	Valida abordagens complexas (caminhos mínimos, grafos ponderados) com feedback granular	Central: distingue corretude lógica de complexidade ótima

O processo de instanciação do *corpus* demonstra que a tipologia do WOKDEX atua como um verdadeiro andaime pedagógico: ela instaura uma gradatividade na correção que é invisível para formatos de empacotamento tradicionais (ICPC, Kattis, Polygon). A Tabela 8 confirma a exclusividade do WOKDEX nessas capacidades.

Tabela 8. Comparativo de capacidades por formato de empacotamento

Capacidade	ICPC	Kattis	Polygon	PExIL	WOKDEX
Teste de I/O simples	✓	✓	✓	✓	✓
Limite de tempo global	✓	✓	✓	✓	✓
Tipo pedagógico do teste	✗	✗	✗	✗	✓
Dica formativa por cenário	✗	✗	✗	✗	✓
Detecção de falso positivo	✗	✗	✗	✗	✓
Separação corretude/performance	✗	✗	✗	✗	✓
Mapeamento de skills	✗	✗	✗	✗	✓

7. Discussão

7.1. Implicações Pedagógicas

O schema WOKDEX transforma a avaliação automática de um modelo de *detecção de erros* para um modelo de *diagnóstico pedagógico*. Essa distinção é significativa pois: (1)

[1]A documentação do *corpus* e os metadados encontram-se disponíveis em `https://alessiojr.github.io/wokdex-corpus/`

erros podem ser corrigidos por tentativa e erro, enquanto *misconceptions* requerem reestruturação cognitiva [Anderson 1983]; (2) o **feedback sobre erros** orienta o que corrigir, mas sobre *misconceptions* orienta o que reaprender; (3) **separar corretude de performance** reduz a frustração de quem possui a lógica certa; e (4) **falsos positivos silenciosos** (MC^3) são os mais danosos, reforçando modelos mentais incorretos [Shute 2008].

7.2. O Gargalo da Autoria e o Conhecimento Pedagógico de Conteúdo (PCK)

O modelo clássico de juízes online "terceiriza" a complexidade da correção para a máquina no momento da submissão. O schema WOKDEX altera esse paradigma, transferindo a complexidade para o momento da autoria. Elaborar cenários `MISCONCEPTION` exige do docente um elevado Conhecimento Pedagógico de Conteúdo (PCK) [Shulman 1986]: ele precisa não apenas dominar o algoritmo ótimo, mas antecipar sistematicamente os modelos mentais errôneos dos estudantes.

Este alto custo cognitivo e de tempo explica a hegemonia histórica de formatos tradicionais (ICPC, Polygon). Contudo, levantamos a hipótese de que o Retorno sobre Investimento (ROI) formativo compensa este esforço, pois o juiz passa a atuar como um tutor assíncrono em escala. Para mitigar esse gargalo na prática, uma plataforma de curadoria (WokManager) permite a edição estruturada dos metadados WOKDEX e potencializa a criação de cenários pedagógicos. Aliada a pipelines de IA generativa (*human-in-the-loop*), essa infraestrutura visa apoiar os professores na ideação de casos de borda baseados em ontologias de erros clássicos [Robins 2019].

7.3. Carga Cognitiva e o Fluxo *Fail-Fast*

A decisão de interromper a avaliação e emitir feedback no primeiro tipo de erro detectado (Figura 1) ancora-se na Teoria da Carga Cognitiva [Sweller 1988]. Caso o sistema relatasse simultaneamente um erro funcional, uma *misconception* e um alerta de performance $O(N^2)$, o estudante novato sofreria sobrecarga. O WOKDEX fornece um andaime progressivo (*scaffolding*): o aluno ganha autoeficácia ao validar sua lógica funcional e, apenas num segundo momento, é instado a otimizar a complexidade de sua solução.

7.4. Validação Retrospectiva *In-Silico*

Para validar empiricamente a eficácia diagnóstica da arquitetura proposta sem depender imediatamente de ensaios *in-vivo*, conduzimos uma avaliação retrospectiva *in-silico*. A partir de um acervo histórico anonimizado de 2.654 submissões legadas (tratadas como dados secundários desidentificados para garantir a conformidade ética), isolamos uma amostra de 856 tentativas que correspondiam exatamente aos 45 exercícios do nosso *corpus* e que haviam recebido **vereditos opacos de reprovação (como *Wrong Answer* e *Time Limit Exceeded*)** no juiz tradicional. Ao reprocessar exclusivamente estas 856 submissões reprovadas através do motor WOKDEX, o "silêncio pedagógico" foi rompido: o sistema reclassificou 235 submissões (27,45%) como cenários de `PERFORMANCE` (revelando que o estudante possuía a corretude lógica e faria jus a uma avaliação intermediária ou nota 'B', mas falhou na eficiência assintótica); e diagnosticou 93 submissões (10,86%) como `MISCONCEPTION`, atingindo o *match* estrito da inversão semântica com alta confiança. As restantes 528 submissões (61,68%) foram categorizadas como erros do tipo `FUNCTIONAL` ou de lógica incompleta, passando a estar mapeadas para receber as dicas formativas. Esta prova de conceito (*Proof of Concept*) demonstra categoricamente a

capacidade do *schema* de resgatar códigos subavaliados e converter métricas binárias em avaliações granulares e justas.

7.5. Ameaças à Validade e Limitações

Para assegurar o rigor metodológico [Wohlin et al. 2012], a Tabela 9 sintetiza as limitações do presente estudo.

Tabela 9. Ameaças à validade e limitações

ID	Ameaça	Descrição e Mitigação
L1	Validade de Constructo (Falsos Diagnósticos)	Falhas no cenário `MISCONCEPTION` nem sempre indicam *misconceptions*; podem ser *slips*. Mitigação: HMD qualifica o feedback ("*Possível causa:*"). Validação futura: protocolos *think-aloud*.
L2	Gamificação Reversa (*Hint Abuse*)	Dicas precisas podem induzir correção por tentativa e erro sem reflexão. Mitigação futura: decaimento da especificidade baseado no volume de submissões.
L3	Validade Externa (Generalização)	Taxonomia validada para paradigmas imperativos e OO (C, Java, Python). Expansão para linguagens funcionais (Haskell) exigirá novos subtipos.
L4	Validação Empírica	Trabalho estabelece o artefato e prova viabilidade curatorial. Eficácia das dicas exige validação em sala de aula (estudo quase-experimental em planejamento).

8. Conclusão

Este artigo apresentou o WOKDEX, um modelo de metadados que mitiga o "silêncio pedagógico" dos juízes online ao injetar a intenção formativa diretamente no artefato do exercício. Diferente de padrões focados apenas em portabilidade técnica (como o ICPC) ou de *autograders* que geram aprisionamento pedagógico (*platform-lock*), a taxonomia proposta — destacando cenários de `MISCONCEPTION` e `PERFORMANCE` — aliada à Heurística de Modulação de Dica (HMD), permite um diagnóstico contextualizado e agnóstico de sistema. A validação empírica *in-silico* comprovou esta eficácia ao resgatar 38,31% dos vereditos opacos (como WA e TLE) em uma amostra de 856 submissões reprovadas, transformando métricas binárias punitivas em avaliações granulares. Em última análise, o modelo estabelece as bases para que plataformas somativas de massa atuem como tutores assíncronos em larga escala. O *corpus* e as especificações encontram-se disponíveis sob os princípios de Ciência Aberta.

Trabalhos futuros concentram-se em: (1) avaliação empírica *in-vivo* comparando turmas com o WOKDEX e turmas de controle, com foco no tempo até o sucesso (*Time-to-Success*) e taxa de falhas; (2) exploração da Teoria da Autorregulação da Aprendizagem para verificar como a HMD afeta a autonomia e mitiga a gamificação reversa; (3) adoção de mecanismos nativos para testes de mutação no *schema*; (4) uso de IA generativa para mitigar o custo de autoria dos cenários; e (5) expansão para paradigmas declarativos e integração arquitetural com o ecossistema LMS (xAPI, IMS LTI, QTI).

Agradecimentos

O autor agradece ao Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) pela concessão de licença integral para o processo de doutoramento.

Declaração sobre uso de Inteligência Artificial

Ferramentas de IA generativa foram utilizadas como assistência na organização e revisão textual deste artigo, conforme o Código de Conduta para Autores da SBC.

Referências

ACM/IEEE (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM Press and IEEE Computer Society Press, New York, NY, USA. .

ACM/IEEE (2020). Computing curricula 2020 (cc2020): Paradigms for global computing education. https://dl.acm.org/doi/epdf/10.1145/3467967. .

ADL Initiative (2004). SCORM 2004 4th edition overview. Technical report, Advanced Distributed Learning (ADL) Co-Lab.

ADL Initiative (2015). Experience API (xAPI) specification, version 1.0.3. Technical report, Advanced Distributed Learning (ADL) Co-Lab.

Alves, F. and Jaques, P. A. (2014). Um ambiente virtual com feedback personalizado para apoio a disciplinas de programação. In *Anais do XXV Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1078–1082. SBC. http://milanesa.ime.usp.br/rbie/index.php/sbie/article/download/3051/2715.

Anderson, J. R. (1983). *The Architecture of Cognition*. Number 5 in Cognitive Science Series. Harvard University Press, Cambridge, MA. .

Beecrowd (2024). Beecrowd: Plataforma de problemas de programação. https://judge.beecrowd.com/.

Black, P. and Wiliam, D. (1998). Assessment and classroom learning. *Assessment in Education: Principles, Policy & Practice*, 5(1):7–74.

Costa, E., Fechine, J., Silva, P., and Rocha, H. J. B. (2016). *Modelos de Feedback para estudantes em Ambientes Virtuais de Aprendizagem*, pages 1–38. https://doi.org/10.5753/sbc.11436.9.1.

de Campos, C. P. and Ferreira, C. E. (2004). Boca: um sistema de apoio a competições de programação. *Journal of Educational Technology*, 1(1):25–35. .

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. *ACM SIGCSE Bulletin*, 36(1):26–30.

Edwards, S. H. and Pérez-Quiñones, M. A. (2008). Web-cat: Automatically grading programming assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, page 328. ACM. https://doi.org/10.1145/1597849.1384371.

Hake, R. R. (1998). Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1):64–74. .

ICPC (2024). Problem package format specification. 2026-03-22 https://icpc.io/problem-package-format/.

Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, page 86–93, New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/1930464.1930480.

IMS Global Learning Consortium (2013). IMS question and test interoperability (QTI) specification, version 2.1. Technical report, IMS Global Learning Consortium.

IMS Global Learning Consortium (2019). IMS learning tools interoperability (LTI) core specification, version 1.3. Technical report, IMS Global Learning Consortium.

Kattis (2016). Kattis problem archive. https://open.kattis.com/.

Keuning, H., Jeuring, J., and Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ.*, 19(1). https://doi.org/10.1145/3231711.

LeetCode (2015). LeetCode: Online coding challenges. https://leetcode.com/.

Lobb, R. and Harlow, J. (2016). Coderunner: a tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51. https://doi.org/10.1145/2810041.

Mariani, J. E., Gerber, N., and Kinkhorst, T. (2005). DOMjudge: An automated judge system for programming contests.

Messer, M., Brown, N. C. C., Kölling, M., and Shi, M. (2024). Automated grading and feedback tools for programming education: A systematic review. *ACM Trans. Comput. Educ.*, 24(1). https://doi.org/10.1145/3636515.

Mirzayanov, M., Pavlova, O., Mavrin, P., Melnikov, R., Plotnikov, A., Parfenov, V., and Stankevich, A. (2020). Codeforces as an educational platform for learning programming in digitalization. *Olympiads in Informatics*, 14:133–142. https://ioi.te.lv/oi/pdf/v14_2020_133_142.pdf.

Paiva, J. C., Leal, J. P., and Figueira, Á. (2022). Automated assessment in computer science education: A state-of-the-art review. *ACM Transactions on Computing Education (TOCE)*, 22(3):1–40. https://doi.org/10.1145/3513140.

Queirós, R. and Leal, J. P. (2012). PExIL: XML language for programming exercises interoperability. *Informatics in Education*, 11(1):95–112.

Robins, A. (2019). Novice programmers and introductory programming. *The Cambridge Handbook of Computing Education Research*, pages 327–376. https://doi.org/10.1017/9781108654555.013.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172. https://doi.org/10.1076/csed.13.2.137.14200.

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational researcher*, 15(2):4–14.

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research*, 78(1):153–189. https://doi.org/10.3102/0034654307313795.

Silva, E. P. d. (2024). *Misconceptions in Correct Code: Assisting Instructors and Students by Shedding Light on What is Potentially Overshadowed by Automated Correction*. Tese de doutorado, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil. https://doi.org/10.47749/T/UNICAMP.2024.1408046.

Striewe, M. and Balz, M. (2015). A standard format for programming exercises. *Formative Assessment*.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, 12(2):257–285.

Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA. https://www.hup.harvard.edu/catalog.php?isbn=9780674576292.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Comput. Surv.*, 51(1). https://doi.org/10.1145/3143560.

Watson, C. and Li, F. W. (2014). Failure rates in introductory programming revisited. *ACM Transactions on Computing Education*, 14(1):1–24. https://dl.acm.org/doi/10.1145/2591708.2591749.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer. https://dl.acm.org/doi/book/10.5555/2349018.