# Ensinando Raciocínio Algorítmico em Tempos de Inteligência Artificial

**Como ensinar pensamento computacional apesar da Inteligência Artificial**

Frank Coelho de Alcantara

2025-06-20

# Sumário

# 1 Resumo

Este estudo propõe a implementação de uma disciplina introdutória de **Raciocínio Algorítmico** para cursos de Ciência e Engenharia da Computação, fundamentada em uma metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração). O objetivo é desenvolver o **Pensamento Computacional** nos alunos, uma habilidade reconhecida acadêmica e globalmente por sua importância para a resolução de problemas complexos no século XXI.

O trabalho explora a integração crescente do **Pensamento Computacional** em currículos de universidades renomadas nos EUA, Reino Unido, Europa e China, destacando uma mudança do foco exclusivo na programação para uma compreensão mais ampla dos princípios computacionais.

Para a prova de conceito, foi criada uma estrutura curricular detalhada para uma disciplina, chamada de **Raciocínio Algorítmico** de 80 horas. Este projeto foi criado com metodologia **DAAD**, a experiência de 20 anos de aulas do autor em Universidades da Cidade de Curitiba. A disciplina está dividida em 4 módulos de ensino com conteúdo e exercícios que progridem do conceitual ao prático, enfatizando a aplicação de cada componente **DAAD**. Além disso, o estudo sugere que essa metodologia pode ser generalizada para outras disciplinas de graduação em ciências, matemática e engenharia. Finalmente este estudo inclui um conjunto de problemas resolvidos como exemplos de aplicação prática da metodologia **DAAD** para cada módulo da disciplina proposta.

As **estratégias pedagógicas** recomendadas incluem aprendizagem ativa, atividades *unplugged* e *plugged*, e oportunidades para o fomento da colaboração. Além disso, o estudo conclui que os processos de avaliação devem ser abrangente, direcionada e, ao mesmo tempo diversificados. Entretanto, a estrutura de avaliação deve ser exclusiva do professor e não faz parte deste estudo.

# 2 Introdução: Pensamento Computacional e a Metodologia DAAD

Está sem tempo? Leia o Resumo da introdução aqui.

Este estudo apresenta uma análise sobre a criação de uma disciplina de **Raciocínio Algorítmico** como prova de conceito para o uso metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração)[1] por professores de graduação em cursos de Ciência e Engenharia da Computação seguindo uma proposta da Universidade de Cambridge (1). Tal disciplina deverá introdutória, primeiro período destes cursos visando criar a estrutura cognitiva necessária ao entendimento da computação e sua aplicação para resolução de problemas. Contudo, o estudo ressalta que a metodologia **DAAD** pode ser aplicada a qualquer disciplina de qualquer curso, e que o **Pensamento Computacional** é uma habilidade essencial para o século XXI. Tal esforço se mostra necessário devido à crescente importância do **Pensamento Computacional** e, consequentemente, do **Raciocínio Algorítmico**, além da popularização do acesso à ferramentas de Inteligência Artificial (2) (3).

A motivação para a proposta de criação desta metodologia surgiu da identificação de lacunas no conhecimento dos alunos de Ciência e Engenharia da Computação, percebidas no estudo dos resultados da últimas avaliações do **E**xame **N**acional de **D**esempenho dos **E**studantes, **ENADE**, dos cursos de Ciência da Computação e Engenharia da Computação. Olhando as últimas edições destes exames e considerando apenas as médias das questões discursivas do componente de conhecimento específico com médias de $9,8$ e $16,9$ de 100 respectivamente [(4)](5). Além disso, parece haver algumas perspectivas conflitantes sobre quais habilidades cognitivas permitem o desenvolvimento eficaz de algoritmos. Principalmente, porque esta é uma questão em aberto debate na comunidade acadêmica.

## 2.1 A Jornada Inacabada: O Problema do Ensino do Raciocínio Algorítmico

As críticas ao ensino de **Raciocínio Algorítmico** aparecem de forma relevante na academia a partir das últimas décadas do século XX. Os estudos pioneiros de Lithner (2008)(6) destacaram que o ensino tradicional prioriza **Raciocínio Algorítmico Imitativo**, no qual alunos reproduzem procedimentos memorizados, em inglês *Familiar Algorithmic Reasoning*, em detrimento do **Raciocínio Matemático Criativo**. Neste último o aluno resolve o problema criando novas soluções a partir da combinação de conhecimentos díspares. Essa abordagem suprime a capacidade de decompor problemas e construir soluções originais, perpetuando uma cultura de superficialidade cognitiva (7). Os trabalhos de Lithner (2008)(6) e Harisman (2023) indicam que o ensino por imitação é parte significativa do problema. Mas, não é a única.

Pesquisas em educação matemática indicam que o ensino baseado em **Raciocínio Algorítmico Imitativo** pode comprometer a originalidade dos estudantes na solução de problemas novos. Isso ocorre porque a dependência de algoritmos predefinidos limita a capacidade de

[1] Algoritmização é um neologismo para traduzir o termo "Algorithmization" do inglês. A ideia é que seja um termo que remete à criação de algoritmos, ou seja, a construção de soluções sistemáticas e eficientes para problemas computacionais. É horrível, eu sei.

adaptação e inovação dos alunos diante de situações inéditas. Conforme Hurrell (2021)(8), o conhecimento procedimental, como fazer, caracterizado por sequências fixas de ações repetidas, não garante a compreensão necessária para gerar novas estratégias ou adaptar as conhecidas a novos problemas, o que reduz a originalidade na resolução.

Além disso, o *National Council of Teachers of Mathematics*, em seus Princípios e Padrões para a Matemática Escolar enfatiza que a aprendizagem matemática deve ir além do domínio de procedimentos algorítmicos e procedurais, promovendo a compreensão conceitual e a flexibilidade cognitiva(9). O documento alerta que a dependência excessiva de algoritmos e regras fixas pode inibir a transferência de conhecimentos para contextos inéditos e a capacidade de resolver problemas de forma criativa e inovadora (9).

Estudos empíricos, ainda mais antigos, indicam que estudantes treinados predominantemente com técnicas algorítmicas apresentam até 32/ menos originalidade na solução de problemas novos, em comparação com aqueles que desenvolvem compreensão conceitual e estratégias flexíveis (10).

Finalmente existem críticas sobre a capacidade e eficiência da compreensão. O livro *Programming: The Derivation of Algorithms* (11) expôs como o ensino imperativo tradicional foca na sintaxe de laços de repetição e invariantes, usando a lógica de Hoare, negligenciando a construção de entendimento conceituais fundamentais. Essa metodologia foi criticada por substituir o "porquê" pelo "como", limitando o pensamento computacional profundo (12). A Table 2.1 resume as principais críticas ao ensino tradicional de **Raciocínio Algorítmico**.

Table 2.1: Estudos e possíveis razões por trás dos problemas persistentes no ensino de **Raciocínio Algorítmico**.

| Fator | Impacto Histórico | Evidência Atual |
|---|---|---|
| **Currículos Baseados em Eficiência** | Priorização de otimização (ex: complexidade O(n)) sobre metacognição. | Algoritmos como *QuickSort* são ensinados como fórmulas pré-definidas, sem discussão ou cognição (12). |
| **Falta de Formação Docente** | Professores reproduzem métodos tradicionais por falta de treino em pedagogia criativa. | Estudos mostram que educadores não dominam técnicas para mitigar dependência de Inteligência Artificial (13) |
| **Infraestrutura Cognitiva** | Modelos mentais baseados em **Raciocínio Algorítmico Imitativo** dificultam a transição para **Raciocínio Matemático Criativo**. | Meta-análises confirmam baixa transferência de conceitos algorítmicos entre domínios [(7)](14) |

Um problema mais recente para a criação das competências que levem ao **Raciocínio Algoritmico** e ao **Pensamento Computacional** parece estar relacionado com a Inteligência Artificial.

O interesse na pesquisa de soluções de Inteligência Artificial que segundo a Figure 2.1 se acentuou a partir de 2010.

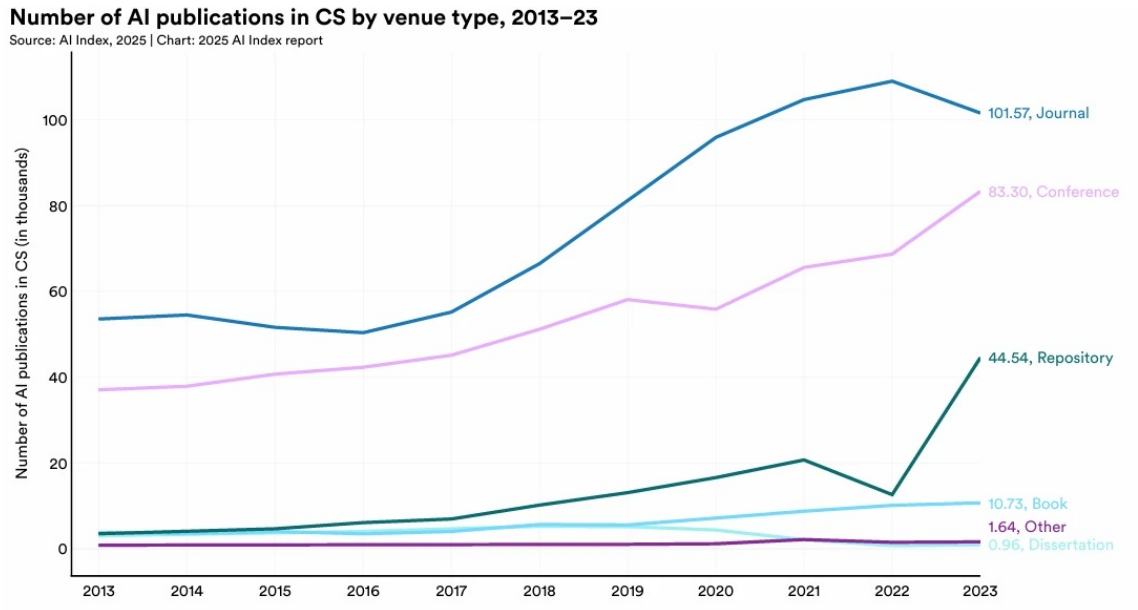


Figure 2.1: Gráfico mostrando a evolução de publicações sobre Inteligência Artificial por tipo de publicação (2).

Este crescimento teve impacto na disponibilidade destas tecnologias, o que pode ser corroborado se consideramos o aumento do uso de dispositivos contendo soluções embarcadas de Inteligência artificial como pode ser visto na Figure 2.2.

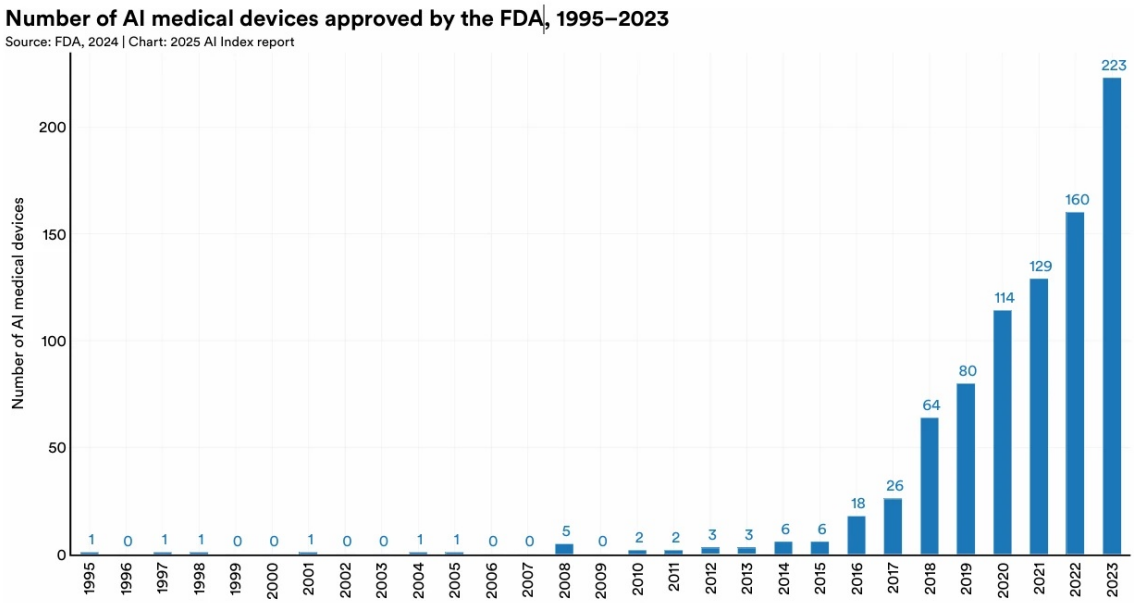


Figure 2.2: Gráfico mostrando a evolução do número de dispositivos médicos contendo Inteligência Artificial embarcada segundo os dados da (2).

Esta velocidade de adaptação parece ter impactos negativos no binômio ensino-aprendizagem de forma geral em todos os cursos e formações.

Um estudo recente conduzido pelo MIT Media Lab, intitulado *Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task*[2], pub-

[2]em tradução livre "Seu Cérebro no ChatGPT: Acúmulo de Dívida Cognitiva ao Usar um Assistente de

licado em junho de 2025, investigou os impactos do uso de modelos de linguagem grandes (LLMs), como o ChatGPT, na cognição humana, especificamente no contexto da escrita de ensaios(15). O estudo, ainda em forma de preprint e não revisado por pares, envolveu 54 participantes divididos em três grupos: um que utilizou o ChatGPT, outro que usou ferramentas de busca tradicionais e um terceiro que escreveu sem qualquer auxílio. Os resultados indicaram que os participantes que utilizaram o ChatGPT apresentaram menor atividade cerebral, menor retenção de memória e menor originalidade em seus escritos em comparação com os outros grupos(15). Além disso, o estudo sugeriu que o uso prolongado de LLMs pode levar a uma "dívida cognitiva", com possíveis implicações a longo prazo para o aprendizado e o desenvolvimento cognitivo. No entanto, devido às limitações do estudo, como o tamanho da amostra e o foco específico no ChatGPT, os achados devem ser interpretados com cautela, e mais pesquisas são necessárias para generalizar os resultados (15).

O estudo de SILVA QUINTO, W. A. et al. (16) investiga o impacto da Inteligência Artificial no desenvolvimento do pensamento crítico entre estudantes de Tecnologia da Informação na Região Norte do Brasil. Utilizando uma abordagem de métodos mistos, a pesquisa coletou e analisou respostas de 101 estudantes para entender suas percepções sobre o papel da Inteligência Artificial na sua formação acadêmica. Os resultados revelam que uma grande maioria (88,) reconhece a influência da Inteligência Artificial em seus estudos, com quase metade ($47,3\%$) acreditando que as ferramentas de Inteligência Artificial facilitam a aprendizagem. No entanto, uma menor porção ($13,2\%$) expressa preocupações de que a Inteligência Artificial pode impedir o desenvolvimento do pensamento crítico(16).

O artigo *Teaching AI with games: the impact of generative AI drawing on computational thinking skills*[3], publicado em 2025 investiga o impacto do uso de ferramentas generativas de Inteligência Artificial, como ferramentas generativas para desenho, no desenvolvimento de habilidades de **Pensamento Computacional** em 56 alunos do sexto ano de escolas no norte de Taiwan(17). Divididos em dois grupos: o experimental usando estas ferramentas com Inteligência Artificial embarcada e programação baseada em blocos para criar jogos; e o grupo de controle controle, usando ferramentas de busca disponíveis na internet. Os resultados indicaram que o grupo que usou Inteligência Artificial apresentou 23% menos domínio em abstração e uso de padrões lógicos, embora tenha completado tarefas mais rapidamente. A pesquisa sugere que a Inteligência Artificial (IA) acelera a execução, mas reduz a cognição lógica(17). Ainda que os autores tenham usado o termo **Pensamento Algorítmico** com sinônimo do conceito que aqui chamamos de **Raciocínio Algorítmico**, é preciso ressaltar que o artigo destaca necessidade de integrar essas ferramentas de forma equilibrada para promover habilidades de estão intimamente relacionadas com o **Pensamento Computacional**.

Nem tudo são críticas, ao longo do tempo foram realizadas tentativas de resolver os problemas do ensino de **Raciocínio Algorítmico**. Duas abordagens que se destacam: a *Constructive Algorithmics*, que em português poderia ser traduzida como algorítmica construtiva, e a Computação *unplugged*.

A abordagem *Constructive Algorithmics*, desenvolvida por Richard Bird e Oege de Moor em 1997@BirdDeMoor1997, fundamenta-se no uso de **raciocínio equacional** e princípios da álgebra de programas por meio de uma abordagem matemática para a construção de programas de computador, tratando a programação como uma disciplina de engenharia e não como uma arte baseada em tentativa e erro(13). Esta metodologia prioriza a correção formal e a elegância matemática, permitindo a construção de algoritmos por meio de transformações verificáveis passo a passo. Um exemplo emblemático é a derivação de algoritmos de ordenação, como *quicksort*, via composição funcional, onde propriedades matemáticas garantem robustez lógica. Jeremy Gibbons 2020@GibbonsHaskell2020 aplicou esses mesmos princípios

Inteligência Artificial para Tarefa de Escrita de Ensaio".

[3]em tradução livre "Ensinando Inteligência Artificial com jogos: o impacto da Inteligência Artificial generativa no desenvolvimento de habilidades de pensamento computacional".

em *Algorithm Design with Haskell*[4].

A computação *unplugged* emerge como contraponto pedagógico, utilizando atividades manuais, como jogos de tabuleiro para decomposição de problemas e atividades com lápis e papel, para desenvolver bases conceituais do pensamento computacional. Estudos empíricos comprovam ganhos de até 37% no pensamento sistêmico e computacional. Os alunos internalizam conceitos abstratos, recursão, paralelismo, por meio de manipulação física e erro reflexivo (18). Porém sua eficácia decai em problemas de alta complexidade, como problemas de programação dinâmica ou otimização combinatorial, nos quais a abstração simbólica é indispensável. Por outro lado, estudos comparativos, como *Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students*[5](19), demonstram que a falta de transição para ferramentas digitais limita a aplicação prática: alunos dominam puzzles com blocos, mas falham em traduzir lógica para código em problemas do mundo real, expondo uma fratura escalar no modelo (20).

A Table 2.2 resume as principais diferenças entre as abordagens *Constructive Algorithmics* e Computação *unplugged*.

Table 2.2: Comparação entre as abordagens *Constructive Algorithmics* e Computação *unplugged*.

| **Abordagem** | **Descrição** | **Pontos Fortes** | **Limitações** |
|---|---|---|---|
| **Algorítmica Construtiva** | Usa raciocínio equacional e álgebra de programas para derivar algoritmos em linguagens funcionais como Haskell (21). | Rigor matemático, correção formal, elegância na derivação de algoritmos (ex.: quicksort). | Descontextualização ética, negligencia impactos sociais. |
| **Computação *Unplugged*** | Utiliza atividades manuais (ex.: jogos de tabuleiro) para ensinar pensamento computacional (19). | Ganhos de até 37% no pensamento sistêmico, internalização intuitiva de conceitos (recursão, paralelismo). | Ineficaz em problemas complexos (ex.: programação dinâmica), limitada transição para ferramentas digitais. |

Da ambiguidade, surge a integração híbrida como caminho promissor: iniciam-se com atividades não digitais, ou eletrônicas, para fundamentos e migra-se para *Constructive Algorithmics* com extensões críticas — como análise de vieses em algoritmos reais. Bird e de Moor em 1997@BirdDeMoor1997 já forneceram base para isso: seu método equacional pode ser estendido enquanto Gibbons (2020)(21) oferece ferramentas para essa transição ao usar Haskell em exemplos aplicados.

Das pesquisas citada é possível inferir que ensino de **Raciocínio Algorítmico** e **Pensamento Computacional** permanece irresoluto porque supervaloriza eficiência em detrimento da crítica: nas décadas de 1980 a 2000, pedagogias formaram gerações capazes de implementar *BubbleSort*, mas não de questionar *por que usá-lo* (21). Hoje, sistemas baseados em Inteligência Artificial agravam a superficialidade cognitiva (17).

Hoje, em julho de 2025, o cenário educacional ainda carece de uma abordagem unificada e estruturada para o ensino do **Pensamento Computacional** e do **Raciocínio Algorítmico**. A falta de uma metodologia clara, eficaz e prática para o ensino desses conceitos fundamentais limita o desenvolvimento das habilidades necessárias para a resolução de problemas

[4]em tradução livre "Projeto de Algoritmos com Haskell".
[5]disponível em: XLogo4Schools.

computacionais. Dessa forma, este estudo propõe a criação de uma disciplina introdutória que utiliza a metodologia **DAAD** como base para o desenvolvimento dessas competências, visando preencher essa lacuna no ensino superior. Uma disciplina que poderá ser chamada de **Raciocínio Algorítmico**.

## 2.2 Primeiro Contato com a Metodologia DAAD

A metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração), aqui proposta, estende uma discussão da Cambridge Mathmatics (1) e a análise feita por LEHMANN (2024)(22), sob os efeitos da algoritmização. O que este estudo propõe difere dos estudos anteriores por apresentar um *framework* prático iterativo de ações, exercícios e temas de estudo com o intuito de permitir a criação de **Pensamento Computacional** e **Raciocínio Algorítmico** em alunos de graduação, baseado nos estudos teóricos e empíricos de Wing (2006)(23), Lehmann (2023, 2024)(24),(22) e KONG, Siu-Cheung et al. (2019)(25) e muitos outros que citados ao longo do trabalho, sobre temas que abordam o **Raciocínio Algorítmmico** e o **Pensamento Computacional**. Para tanto, este *framework* está estruturado em quatro estágios iterativos:

1. **Decomposição (D)**: processo de fragmentação de problemas complexos em subproblemas gerenciáveis, seguindo princípios de divisão funcional ou estrutural. Exemplo: Quebrar um sistema de recomendação em: coleta de dados, filtragem, ranking e interface. Wing (2006)(23), SBC (2017)(26) e Lehmann (2024)(22) estabelecem decomposição como pilar cognitivo essencial.

2. **Abstração (A)**: identificação seletiva de padrões e invariantes essenciais, com descarte consciente de detalhes irrelevantes ao núcleo do problema. Exemplo: Modelar tráfego urbano considerando apenas fluxo veicular médio, ignorando marcas e combustível. Baseia-se no conceito de "abstração progressiva" perceptível na Linguagem LOGO de Papert (1985)(27).

3. **Algoritmização (A)**: formulação de soluções por meio de sequências lógico-operacionais, garantindo completude e implementabilidade. Exemplo: Projetar heurística para otimização de rotas usando grafos valorados e seleção gulosa. Alinha-se ao paradigma de descrição de estados finais. A essência do projeto algorítmico reside na descrição formal de estados finais, onde a corretude é verificada pelo atendimento inequívoco de pós-condições [(28)](13).

4. **Depuração (D)**: refinamento iterativo mediante validação empírica, incluindo testes de robustez, análise de falhas e otimização pós-implementação. Exemplo: Injeção de dados corrompidos em *pipelines* de processamento para validar resiliência. Incorpora princípios de "engenharia resiliente" de Leveson (2012)(29).

Em um ambiente no qual existem ambiguidades e críticas sobre a formação fundamental do dos alunos de graduação, a metodologia **DAAD** foca prioritariamente nos processos matemáticos e lógicos envolvidos na criação e depuração de soluções computacionais como ferramentas indispensáveis para a solução de problemas.

A busca bibliográfica realizada para este estudo indicou que a metodologia **DAAD** como está proposta parece ser a única ferramenta disponível que oferece uma estrutura prática iterativa para o ensino do **Raciocínio Algorítmico**, abordando tanto a decomposição de problemas quanto a abstração necessária para a criação de algoritmos eficazes e, principalmente, depuração. A relação e integração conceitual entre os componentes da metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração) está ilustrada na Figure 2.3

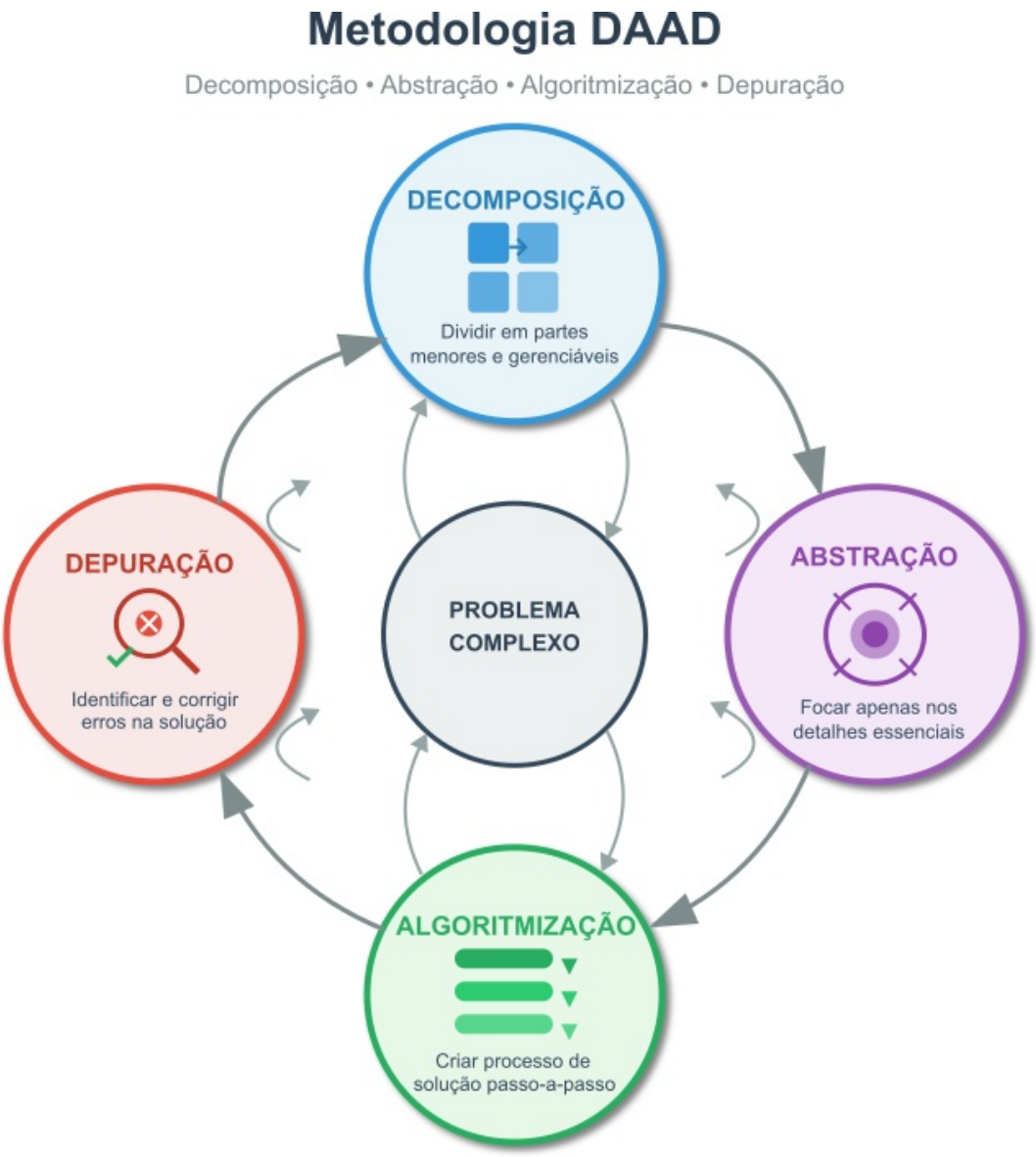


Figure 2.3: Ilustração da inter-relação entre Decomposição, Abstração, Algoritmização e Depuração. fonte:o autor.

O **Pensamento Computacional** é um termo amplo que abrange não apenas a criação de algoritmos, mas também a compreensão de como os computadores processam informações e resolvem problemas (23). Neste contexto, **Raciocínio Algorítmico**, uma parte fundamental do **Pensamento Computacional**, a habilidade de desenvolver instruções passo a passo para resolver problemas, generalizando processos e antecipando impactos de cada etapa (24), criando soluções sistemáticas e eficientes. Contudo, em português parece existir uma fronteira muito tênue entre o que se quer dizer com **Pensamento Computacional** e **Raciocínio Algorítmico**.

Em sua pesquisa RIBEIRO, L. et al (2017)(30) diferenciam **raciocínio lógico**, fundamentado em premissas e inferências, do **raciocínio computacional**, focado em *abstração*, *automação* e *análise* enquanto criticam a redução do **Pensamento Computacional** à criação de algoritmos, ressaltando que o **Raciocínio Algorítmico** é apenas um dos pilares do **Pensamento Computacional**, não sua totalidade. KONG, S. et al.@Kong2019 destacam

que o **Raciocínio Algorítmico** é frequentemente confundido com o **Pensamento Computacional** por ser seu elemento mais tangível, especialmente em contextos de programação. Mais tarde estes mesmos autores alertam que essa visão reduz o **Pensamento Computacional** a habilidades técnicas, negligenciando pilares como a decomposição de problemas e o pensamento crítico(31). O estudo de MEDEIROS (2024)(32) analisou 38 trabalhos brasileiros sobre **Pensamento Computacional** (2018–2024) e identificou que 76% deles equiparavam **Pensamento Computacional** ao ensino de algoritmos, via Scratch ((33)) ou aplicações de robótica. Concluindo que a ênfase excessiva no **Raciocínio Algorítmico** subestima outros pilares do **Pensamento Computacional**, tais como pensamento algébrico e resolução criativa de problemas (26).

Uma questão interessante surge do trabalho de Hora (2022)(34) que destaca que no Brasil o **Pensamento Computacional** é frequentemente restrito a disciplinas de exatas, Matemática e Computação, nas quais o **Raciocínio Algorítmico** domina as práticas pedagógicas criticando o viés tecnicista que ignora dimensões como abstração contextualizada e crítica sociotécnica. Destacando a abrangência do **Pensamento Computacional**. Dessa forma Hora (2022)(34) sustenta os trabalhos de Wing (2006)(23) e (24). Além disso, Felipussi e Padua (2023)(35) perceberam que os professores tendem a associar Pensamento Computacional diretamente à programação de robôs, tratando algoritmo como sinônimo de **Pensamento Computacional**.

A própria Sociedade Brasileira de Computação (2017)(26) adverte que a **BNCC** (**B**ase **N**acional **C**omum **C**urricular) não distingue claramente **Raciocínio Algorítmico** e **Pensamento Computacional**, gerando ambiguidade em propostas pedagógicas e recomenda equilibrar o ensino de algoritmos com atividades de **decomposição de problemas não lineares**.

Finalmente, existem trabalhos que indicam que a ambiguidade entre **Pensamento Computacional** e **Raciocínio Algorítmico** pode ter consequências negativas na formação do indivíduo.

Do trabalho de Medeiros (2024)(32) podemos inferir que os estudantes desenvolvem habilidades técnicas (ex.: codificar em blocos) mas falham em aplicar **Pensamento Computacional** em contextos interdisciplinares. As humanidades e artes raramente integram **Pensamento Computacional** em seus currículos, pois o **Raciocínio Algorítmico** é visto como não aplicável (36). Por fim, as avaliações de **Pensamento Computacional** focam em correção algorítmica e eficiência do código, não em processos mentais como abstração (25).

É necessário considerar que o termo **Pensamento Computacional** foi popularizado por Jeannette Wing, em 2006@Wing2006, argumentando que o **Pensamento Computacional** deveria ser considerado uma habilidade indispensável para todos, comparável à leitura, escrita e aritmética. Conceito no qual ela é apoiada por Saidin (2021)(37) e pela *Computer Science Teachers Association*@CSTA2011a. O **Pensamento Computacional** é reconhecido globalmente como uma das habilidades vitais para o século XXI, essencial para a resolução de problemas complexos em diversas áreas (38). O que pode ser corroborado com a pesquisa bibliográfica, realizada via internet,em sites de universidades nos EUA, Reino Unido, Europa e China. Esta pesquisa demonstrou uma tendência crescente na integração do **Pensamento Computacional** nos currículos de graduação, com ênfase em abordagens práticas e baseadas em projetos cujas características veremos na Chapter 3.

A metodologia **DAAD**, modifica os tradicionais "Quatro Pilares do Pensamento Computacional" de Wing@Wing2006 ao substituir "Reconhecimento de Padrões" por "Depuração" concordando com Lehmann@Lehmann2024 e com a Cambridge Mathematics@CambridgeMaths. Essa mudança adequa o **Pensamento Computacional** ao **Raciocínio Algorítmico** enfatizando o processo ativo de criação de algoritmos e a habilidade crítica de detecção e correção de erros, a depuração. Criando uma metodologia iterativa adequada a solução de problemas em um ciclo de entendimento, abstração, construção da solução e depuração.

## 2.3 A Disciplina de Raciocínio Algorítmico

O estudo propõe uma estrutura curricular detalhada para uma disciplina de 80 horas, dividida em módulos que progridem do entendimento conceitual com a aplicação prática de cada componente **DAAD**. Com a expectativa de que esta metodologia seja aplicada a todas as disciplinas dos cursos de graduação de Ciência e Engenharia da Computação, e generalizada para outros cursos relacionados a ciência, matemática e engenharia.

As estratégias pedagógicas recomendadas neste estudo incluem sala de aula invertida [BergmannSams2012], aprendizagem ativa(39), atividades *unplugged* e *plugged*[6], e fomento da colaboração, visando aprofundar a compreensão e a capacidade de transferência de habilidades.

O estudo dos métodos de avaliação fomenta a necessidade de um estudo posterior. Porém considerando os textos sobre **Pensamento Computacional** utilizados para construção do referencial teórico deste estudo é possível indicar que a avaliação desta disciplina deve ir além da correção do código, utilizando rubricas detalhadas para medir o crescimento das competências **DAAD**(41). Além disso, existem desafios, relacionados a preparação do corpo docente e o engajamento dos alunos que serão abordados apenas com a indicação de melhores práticas, destacando a necessidade de desenvolvimento profissional contínuo e a conexão do conteúdo com problemas do mundo real(42).

Este trabalho está dividido em cinco partes principais e um conjunto de dez apêndices com propostas de atividades para uso em sala e para estudo complementar distribuídas de forma a atender as necessidades **DAAD**. As partes principais são: esta introdução; Chapter 3, na qua estão apresentadas os conceitos **DAAD** e o **Pensamento Computacional** em universidades dos EUA, Reino Unido, Europa e China; Chapter 5 discutindo as estratégias pedagógicas sugeridas na literatura, Chapter 4 na qual estão definidos os princípios da metodologia; Chapter 6, na qual está apresentado o projeto de uma disciplina de 80 horas com a metodologia **DAAD**, em dez subseções; e, por fim, Chapter 7, na qual estão apresentadas as conclusões do trabalho e recomendações para a implementação da disciplina.

[6] *Unplugged* e *plugged* são termos usados para descrever atividades que não usam tecnologia digital, *unplugged*, e aquelas que usam tecnologia digital, *plugged*.

# 3 Analisando Currículos Internacionais

A integração do **Pensamento Computacional** nos currículos do ensino superior tem sido uma tendência crescente, com universidades em todo o mundo revendo os conteúdos e estruturas dos seus cursos e disciplinas para incorporar esses conceitos (WING, 2006). Na maior parte das universidades estudadas, o objetivo é incluir o **Pensamento Computacional** e, consequentemente, o **Raciocínio Algorítmico** além das disciplinas de programação (LI et al, 2020). **Esta evolução curricular representa uma transição de um modelo que ensinava primariamente programar para o computador para um que enfatiza o pensar computacionalmente. Instituições de prestígio estão se afastando de um foco exclusivo na sintaxe de programação para priorizar uma compreensão mais profunda de como os sistemas computacionais funcionam e como aplicar o raciocínio computacional para resolver problemas complexos**(CURZON et al., 2019). Indicando uma maturidade crescente do campo da educação em ciência da computação, na qual a alfabetização computacional é vista como uma competência mais ampla e conceitual, que transcende a mera proficiência numa linguagem de programação específica (WING, 2006).

A comparação entre o ensino no Brasil com universidades internacionais é complexo e desafiador.

Existem diferenças de metodologia entre as universidades da Europa, EUA, China e Brasil. A forma como o tempo de aula presencial, trabalhos e pesquisa é contabilizado e considerado para validação da disciplina tem impacto direto no formato dessas disciplinas para inclusão da Metodologia **DAAD** (**D**ecomposição, **A**bstração, **A**lgoritmização e **D**epuração).

Nos EUA, o sistema predominante é o de ***Credit Hours*** ou ***Semester Credit Hours***. Uma hora de crédito engloba, normalmente, tanto o tempo despendido em sala de aula quanto o estudo independente. Geralmente, **cada hora de crédito corresponde a uma hora de instrução semanal em sala de aula, complementada por duas horas de estudo individual, ao longo de um semestre que dura aproximadamente 15 semanas**. Isso significa que uma disciplina comum de 3 créditos terá aproximadamente 3 horas de aula por semana e 6 horas de estudo fora da sala, totalizando cerca de 45 horas de contato direto com o professor ao longo do semestre e 90 horas de estudo fora de sala. Para a obtenção de um diploma de bacharelado, os estudantes normalmente precisam acumular entre 120 e 130 horas de crédito no curso (NCES, 2010). A Table 3.1 resume este conceito.

Table 3.1: Exemplo de Cálculo de horas, considerando o Sistema de Créditos Acadêmicos nos EUA.

| Créditos da Disciplina | Horas em Sala de Aula (por semestre) | Horas de Estudo Fora da Sala (por semestre) | Horas Totais de Estudo (por semestre) |
|---|---|---|---|
| **1 crédito** | 15 horas | 30 horas | 45 horas |
| **2 créditos** | 30 horas | 60 horas | 90 horas |
| **3 créditos** | 45 horas | 90 horas | 135 horas |

Por outro lado, na Europa, a maioria das instituições acadêmicas adota o Sistema Europeu de Transferência e Acumulação de Créditos, em inglês **E**uropean **C**redit **T**ransfer and **A**ccumulation **S**ystem, **ECTS**. É um sistema que prioriza o volume de trabalho total que um

estudante precisa dedicar para alcançar os resultados de aprendizado esperados (EUROPEAN COMMISSION, 2015). Isso abrange não somente as horas de aula, mas também atividades como estudo autônomo, preparação para avaliações, desenvolvimento de projetos e estágios. O padrão estabelece que 60 créditos **ECTS** equivalem ao volume de trabalho de um estudante em tempo integral durante um ano acadêmico. A conversão de 1 crédito **ECTS** geralmente representa entre 25 e 30 horas de trabalho total do estudante, embora essa proporção possa ter pequenas variações entre os países europeus. Diferentemente do modelo americano, não há uma correlação direta entre os créditos **ECTS** e as horas de contato em sala de aula, pois o foco está no esforço total do estudante, o que pode resultar em disciplinas com muitos créditos **ECTS** e poucas horas de aula se o trabalho independente for intenso. O **ECTS** foi concebido para otimizar a mobilidade estudantil e o reconhecimento de qualificações entre as universidades europeias. Um programa de bacharelado com duração de 3 anos geralmente corresponde a 180 **ECTS**, enquanto um de 4 anos pode totalizar 240 **ECTS** (EUROPEAN COMMISSION, 2015). A Table 3.2 resume este conceito.

Table 3.2: Exemplo de Cálculo de horas, considerando o Sistema Europeu de Transferência e Acumulação de Créditos (**ECTS**).

| Créditos **ECTS** da Disciplina | Horas Totais de Trabalho do Estudante (por semestre/módulo) |
|---|---|
| **1** ECTS**** | 25-30 horas |
| **2** ECTS**** | 50-60 horas |
| **3** ECTS**** | 75-90 horas |

As universidades chinesas geralmente adotam um sistema de créditos que está, na maioria das vezes, vinculado às **horas de contato** semanais. Embora não exista um sistema de crédito totalmente unificado em todas as instituições de ensino superior chinesas, um padrão comum é que **1 crédito corresponda à uma hora de aula por semana**, um hora aula corresponde a 45 minutos, ao longo de um semestre (XIAMEN UNIVERSITY, [s.d.])(SICHUAN UNIVERSITY, 2020).

A duração de um semestre na China é de aproximadamente 18 semanas. Portanto, uma disciplina de 1 crédito pode implicar 18 horas de contato direto em sala de aula por semestre. Assim como nos sistemas ocidentais, espera-se que os alunos dediquem tempo adicional significativo para estudo independente e preparação fora da sala de aula para cada hora de contato. Entretanto, não foi possível encontrar uma relação explícita como nas universidades dos EUA (XIAMEN UNIVERSITY, [s.d.])(SICHUAN UNIVERSITY, 2020).

Para fins de mobilidade estudantil e reconhecimento internacional, algumas universidades chinesas estabelecem equivalências. Por exemplo, a **Xiamen University** (XIAMEN UNIVERSITY, [s.d.]) indica que "Um crédito equivale a uma hora de aula (45 minutos) por semana durante um semestre de cerca de 18 semanas". Outras instituições podem alinhar seus créditos a sistemas internacionais, com algumas conversões indicando que 1 crédito chinês pode ser equivalente a 1.5 a 2 **ECTS** ou aproximadamente 0.5 a 0.67 créditos dos EUA, dependendo da instituição e da especificidade do programa. Novamente há um resumo deste sistema na Table 3.3.

Table 3.3: Resumo do sistema de créditos adotado nas universidades chinesas, com base na carga horária de aulas.

| Créditos na China | Horas de Contato em Sala de Aula (por semestre) | Equivalência **ECTS** (aproximada)* |
|---|---|---|
| **1 crédito** | 18 horas | 1.5 - 2 **ECTS** |

| Créditos na China | Horas de Contato em Sala de Aula (por semestre) | Equivalência **ECTS** (aproximada)* |
|---|---|---|
| **2 créditos** | 36 horas | 3 - 4 **ECTS** |
| **3 créditos** | 54 horas | 4.5 - 6 **ECTS** |

No Brasil, o sistema de créditos nas universidades públicas e privadas é predominantemente baseado na **carga horária de aulas** teóricas e práticas. A **L**ei de **D**iretrizes e **B**ases da Educação Nacional, **LDB** - Lei nº 9.394/1996, (BRASIL, 1996) estabelece a regulamentação geral para a educação superior no país. Tradicionalmente, a conversão mais comum é que **1 crédito equivale a 15 horas-aula**. Contudo, essa equivalência pode variar um pouco entre as instituições e tipos de atividades, geralmente temos:

- Para disciplinas teóricas, 1 crédito geralmente corresponde a 15 horas de aula.
- Para disciplinas práticas, estágios ou atividades de laboratório, a proporção de horas-aula por crédito pode ser diferente, frequentemente exigindo mais horas de contato para 1 crédito (por exemplo, 30 horas de prática para 1 crédito).

O **Ministério da Educação (MEC)**, por meio de resoluções do Conselho Nacional de Educação, define cargas horárias mínimas para a integralização de diferentes cursos de graduação. Essas cargas horárias são, então, desdobradas pelas universidades em disciplinas com seus respectivos créditos. Um ponto relevante é que a **LDB**, no artigo 47, exige um **mínimo de duzentos dias letivos** anuais e, nas instituições públicas de educação superior (BRASIL, 1996).

Nas universidades brasileiras, a determinação de cargas horárias e créditos é um sistema de duas camadas, combinando diretrizes nacionais com a autonomia de cada instituição. O **M**inistério da **E**ducação e **C**ultura, **MEC**, e o **C**onselho **N**acional de **E**ducação, **CNE**, estabelecem a base legal para a carga horária dos cursos de graduação, principalmente através de resoluções e pareceres.

Para cada curso de bacharelado, o **MEC** define uma carga horária total mínima que precisa ser cumprida para que o diploma seja válido. A Resolução CNE/CES nº 2 (BRASIL, 2007), por exemplo, agrupa os cursos em diferentes faixas de carga horária. Um curso de Administração tem no mínimo 3.000 horas, enquanto um curso de Direito exige no mínimo 3.700 horas. A carga horária não se resume apenas ao tempo em sala de aula. Ela inclui as atividades que compõem o trabalho acadêmico do estudante. Conforme o Parecer **CNE/CES** nº 261 (BRASIL, 2006), a hora-aula é um conceito amplo, e a carga horária total do curso pode abranger: aulas teóricas e práticas; horas de estudo individual ou em grupo; trabalhos de campo e de laboratório; estágios supervisionados; atividades de extensão e trabalho de conclusão de curso. As diretrizes do **CNE** também estabelecem um tempo mínimo e máximo para a integralização do curso, garantindo flexibilidade para o aluno, mas também um padrão nacional.

O crédito é, essencialmente, uma unidade de medida do trabalho do estudante. A maioria das universidades brasileiras adota um sistema em que um número de horas de atividades corresponde a um crédito. Em linhas gerais, cada universidade define em seu regimento a equivalência entre horas e créditos. No modelo mais comum, um crédito pode equivaler a 15 horas de atividades em sala de aula. Assim, uma disciplina de 60 horas/aula ao longo de um semestre valeria 4 créditos. Para atividades práticas ou de laboratório, a proporção pode ser diferente. Por exemplo, 30 horas de atividades práticas podem equivaler a 1 crédito (FRAUCHES, 2011). Imagine uma disciplina com a seguinte estrutura em uma universidade hipotética:

- **Carga horária total**: 90 horas;

- **Aulas teóricas**: 60 horas;
- **Aulas práticas**: 30 horas;

Se a universidade define que 1 crédito teórico é igual a 15 horas e 1 um crédito prático corresponde a 30 horas, essa disciplina teria:

- $60/15 = 4$ créditos teóricos
- $30/30 = 1$ crédito prático
- Total de 5 créditos para a disciplina.

Enquanto a carga horária total é regulada nacionalmente, a forma como essa carga é dividida e gerenciada é uma decisão de cada universidade. Não existe uma regra única do **MEC**. Desta forma, as universidades particulares, notadamente as que tem curso superior noturno, ou não consideram horas de trabalho independente, ou não abordam estes trabalhos nas avaliações. A **?@tbl-diff1** permite a comparação entre as disciplinas avaliadas neste estudo e o equivalente em horas aulas no Brasil. Considerando que todo esforço de aprendizado seja feito em sala e apenas as universidades utilizadas neste estudo.

Table 3.4: Comparação entre as disciplinas relacionadas com pensamento computacional e raciocínio algoritmo e a carga horária no Brasil.{#tbl-diff1}

| País | Instituição | Disciplina | Créditos Originais | Horas-Aula Brasil* |
|---|---|---|---|---|
| **Estados Unidos** | Carnegie Mellon University | Princípios da Computação | 10 créditos US | **150 horas-aula** |
| **Estados Unidos** | Carnegie Mellon University | Estrutura de Dados e Algoritmos | 12 créditos US | **180 horas-aula** |
| **Estados Unidos** | Carnegie Mellon University | Codificação e Pensamento Computacional com VEX V5 | Curso de extensão | Curso de extensão |
| **Estados Unidos** | MIT | Introdução ao Pensamento Computacional em Python (6.100A) | 12 créditos US | **180 horas-aula** |
| **Estados Unidos** | MIT | Introdução ao Pensamento Computacional e Ciência de Dados (6.100B) | 12 créditos US | **180 horas-aula** |
| **Estados Unidos** | Penn State University | Disciplinas de Ciência da Computação (integradas) | N/D | N/D |
| **Estados Unidos** | University of Wisconsin-Madison | Resolução de Problemas Usando Computadores (COMP SCI 310) | 3 créditos US | **45 horas-aula** |

| País | Instituição | Disciplina | Créditos Originais | Horas-Aula Brasil* |
|---|---|---|---|---|
| **Reino Unido** | Cambridge University | Algoritmos 1 | Parte do curso geral | Parte do curso geral |
| **Reino Unido** | Cambridge University | Computational Thinking Challenge | Projeto | Projeto |
| **Reino Unido** | Cambridge University | Lógica e Algoritmos | Parte do curso geral | Parte do curso geral |
| **Reino Unido** | Oxford University | Ciência da Computação | N/D | N/D |
| **Reino Unido** | Oxford University | Matemática e Ciência da Computação | N/D | N/D |
| **Reino Unido** | Oxford University | Ciência da Computação e Filosofia | N/D | N/D |
| **Reino Unido** | Imperial College London | Matemática Discreta | 7,5 ECTS | **83 horas-aula** |
| **Reino Unido** | Imperial College London | Matemática, Lógica e Raciocínio | 7,5 ECTS | **83 horas-aula** |
| **Reino Unido** | Imperial College London | Grafos e Algoritmos | 7,5 ECTS | **83 horas-aula** |
| **Reino Unido** | Imperial College London | Raciocínio Simbólico (optativa) | 6 ECTS | **66 horas-aula** |
| **Reino Unido** | Imperial College London | Técnicas Computacionais (optativa) | 6 ECTS | **66 horas-aula** |
| **Reino Unido** | Cardiff University | Pensamento Computacional (CM1101) | 20 créditos UK | **200 horas-aula** |
| **Reino Unido** | Cardiff University | Resolução de Problemas com Python | 10 créditos UK | **100 horas-aula** |
| **Reino Unido** | Cardiff University | Programação Orientada a Objetos com Java | 20 créditos UK | **200 horas-aula** |
| **Escócia** | Edinburgh University | Introdução a Computação | 20 créditos UK | **200 horas-aula** |
| **Escócia** | Edinburgh University | Programação de Computadores | 20 créditos UK | **200 horas-aula** |
| **Escócia** | Edinburgh University | Como Resolver Problemas Usando Computadores | N/D | N/D |
| **Suíça** | ETH Zurich | Pensamento Computacional | N/D | N/D |

| País | Instituição | Disciplina | Créditos Originais | Horas-Aula Brasil* |
|---|---|---|---|---|
| **Suíça** | ETH Zurich | Introdução a Programação | 8 ECTS | **88 horas-aula** |
| **Suíça** | ETH Zurich | Estruturas de Dados e Algoritmos | 8 ECTS | **88 horas-aula** |
| **Suíça** | ETH Zurich | Projeto Digital e Arquitetura de Computador | 6 ECTS | **66 horas-aula** |
| **Alemanha** | Technical University of Munich | Pensamento Computacional (pesquisa) | N/D | N/D |
| **Suíça** | École hôtelière de Lausanne (EHL) | Pensamento Computacional | 3,5 ECTS | **30 horas-aula** |
| **China** | Universidade de Shandong | Introdução à Computação e Design de Programas | 4,5 créditos chineses | **81 horas-aula** |
| **China** | Universidade de Shandong | Pensamento Computacional e Programação | 5 créditos chineses | **90 horas-aula** |
| **China** | Universidade de Pequim (PKU) | Fundamentos da Linguagem Python e Aplicações | 3,5 créditos chineses | **63 horas-aula** |
| **China** | Universidade de Pequim (PKU) | Pensamento Computacional | 3 créditos chineses | **54 horas-aula** |
| **China** | Universidade de Tsinghua | Especialização em Estruturas de Dados e Algoritmos | Coursera | Coursera |

Na **?@tbl-diff1** N/D significa Não Disponível. Foram usadas as segintes fórmulas de conversão: - **EUA → Horas-aula:** Créditos US × 15 horas
- **ECTS → Horas-aula:** ECTS × 11 horas (média europeia de contato) - **Reino Unido → Horas-aula:** Créditos UK × 10 horas - **China → Horas-aula:** Créditos China × 18 horas

Além das diferenças na expectativa de estudo independente por parte dos alunos, que podem influenciar o resultado das disciplinas, existem diferenças importantes na forma como as universidades integram o **Pensamento Computacional** e o **Raciocínio Algorítmico** em seus currículos. A seguir, são apresentados exemplos de como universidades de diferentes regiões do mundo estão implementando o **Pensamento Computacional**, com foco na descoberta de elementos **DAAD** nas suas metodologias pedagógicas.

## 3.1 Universidades nos Estados Unidos

A Carnegie Mellon University (CMU) é uma instituição pioneira na promoção do **Pensamento Computacional**, com Jeannette Wing como figura central nesta iniciativa. A

universidade possui um centro específico para **Pensamento Computacional** dedicado (CARNEGIE MELLON, [s.d.]). A abordagem da CMU envolve ***PR**oblem-**O**riented **B**usiness **E**xplorations*, **PROBES**s, uma sigla em inglês para exploração de problemas orientados a negócio. Alunos quer resolvem estes problemas aplicam conceitos computacionais inovadores a problemas práticos para demonstrar o valor do **Pensamento Computacional** (CARNEGIE MELLON, [s.d.]). A disciplina Princípios da Computação (10 Créditos), introduzida em 2005, foca no estudo do processo de computação, não necessariamente na operação de um computador (WING, 2006). A disciplina Estrutura de Dados e Algoritmos (12 Créditos) espera que os alunos aprendam a decompor problemas. Além disso, busca desenvolver habilidades para acompanhar o progresso da solução, avaliar a correção do código e corrigir falhas por meio de depuração automatizada (Guzdial, 2015 apud KIM, 2021).

Em disciplinas diretamente relacionadas com Ciência e Engenharia da Computação, a disciplina Codificação e **Pensamento Computacional** com VEX V5 (curso de extensão) utiliza atividades de programação estruturadas em contextos de projetos do mundo real, ensinando explicitamente a decomposição para simplificar a codificação complexa (CARNEGIE MELLON ROBOTICS ACADEMY, [s.d.]). A ênfase da CMU na aplicação do **Pensamento Computacional** para resolver problemas do mundo real transcende a mera compreensão. "Pensamento Computacional significa pensar algoritimicamente e com a habilidade de aplicar conceitos matemáticos, tais quais indução para desenvolver soluções mais eficientes, justas e seguras" (CARNEGIE MELLON CENTER OF COMPUTATIONAL THINKING, [s.d.]). Essa abordagem cria uma relação causal: problemas autênticos fornecem a motivação e o contexto para que os alunos se engajem profundamente, desenvolvam e apliquem as habilidades **DAAD**.

O MIT, Massachusetts Institute of Technology, reconhece o **Pensamento Computacional** como um tipo distinto de raciocínio rigoroso de importante valor intelectual. Este raciocínio requer e desenvolve modos importantes de comunicação e a necessidade de compreender o impacto transformador da computação em outras disciplinas (DENNIN & TEDRE, 2019). O MIT recomenda um requisito mínimo de computação para todos os estudantes de graduação, enfatizando que o Pensamento Computacional é mais amplo do que a proficiência em programação (MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2017)(MASSACHUSETTS INSTITUTE OF TECHNOLOGY, [s.d.]). As disciplinas Introdução ao Pensamento Computacional em Python (6.100A, 12 Créditos) e Introdução ao Pensamento Computacional e Ciência de Dados (6.100B, 12 Créditos), que utilizam Python como linguagem de programação e visam dar aos alunos a confiança para escrever pequenos programas úteis, independentemente de sua área principal de estudos (MASSACHUSETTS INSTITUTE OF TECHNOLOGY, [s.d.]).

A **Penn State** integra o **Pensamento Computacional** em seu currículo do curso de Ciência da Computação (PENN STATE UNIVERSITY, [s.d.]ab). As disciplinas enfatizam a conceituação e implementação de soluções computacionais, o raciocínio sobre problemas em múltiplos níveis de abstração e a análise de código quanto ao comportamento, eficiência e correção (PENN STATE UNIVERSITY, [s.d.]a). Também abordam habilidades de desenvolvimento e manutenção de programas, como depuração e teste. Já na **University of Wisconsin-Madison** a disciplina Resolução de Problemas Usando Computadores (COMP SCI 310, 3 créditos) introduz a abstração e decomposição de problemas, discute métodos de uso de computadores para resolver problemas, incluindo técnicas elementares de programação, linguagens de manipulação simbólica e pacotes de software. (THE UNIVERSITY OF WISCONSIN-MADISON, [s.d.]a). Finalmente, a **University of Texas at Austin** foca em conceitos fundamentais de Ciência da Computação, incluindo desenvolvimento de algoritmos, decomposição de problemas e técnicas de depuração em sua disciplina Programação Introdutória (créditos não disponíveis) (THE UNIVERSITY OF TEXAS AT AUSTIN, 2015).

## 3.2 Universidades no Reino Unido

Na **Cambridge University** a formação do **Raciocínio Algorítmico** parece ter sido atribuída ao Cambridge Mathematics. Um instituto criado pela união da editora e as faculdades de educação e matemática da Universidade de Cambridge. Nos documentos gerados pelo Cambridge Mathematics (CAMBRIDGE MATHS, [s.d.]) é possível observar uma relação profunda entre o **Pensamento Computacional** e o pensamento funcional. O instituto produz, cursos, seminários e material de aula para ajudar a formação destas competência em alunos de cursos de graduação diferentes. e integrados no currículo, com ênfase em abstração, lógica, algoritmos e representação de dados. Considerando especialmente as disciplinas introdutórias de cursos relacionados a computação teremos: a disciplina Algoritmos 1 (parte do curso geral), que inclui tópicos como ordenação, análise de complexidade, paradigmas de design (divisão e conquista, programação dinâmica, algoritmos gulosos), estruturas de dado, e análise formal da eficiência dos algoritmos(UNIVERSITY OF CAMBRIDGE, [s.d.]a); o desafio *Computational Thinking Challenge*, um projeto de pesquisa e avaliação digital da faculdade de edução, focado em avaliação de competências de **Pensamento Computacional* voltado para estudantes e educadores (UNIVERSITY OF CAMBRIDGE, [s.d.]b); e a disciplina Lógica e Algoritmos (parte do curso geral) que explora métodos formais, lógica proposicional e de predicados, e sua aplicação em algoritmos e verificação de sistemas computacionais(UNIVERSITY OF CAMBRIDGE, [s.d.]c).

A **Oxford University** oferece cursos de Ciência da Computação, Matemática e Ciência da Computação, e Ciência da Computação e Filosofia. Esses programas, desde o início, estimulam o **Pensamento Computacional** por meio do design de programas de computador (UNIVERSITY OF OXFORD, [s.d.]a). **O departamento enfatiza a leitura de materiais que ilustram conceitos de alto nível da ciência da computação**, a motivação por trás deles e sua aplicação (UNIVERSITY OF OXFORD, [s.d.]a). Além disso, a Oxford University está envolvida com o "UK Bebras Challenge", um desafio de **Pensamento Computacional** que visa introduzir o conceito aos alunos nos níveis de ensino anterior à graduação (UK BEBRAS, [s.d.]a)(UNIVERSITY OF OXFORD, [s.d.]d). Finalmente, o *Oxford University Computing Challenge*, em inglês desafio de computação da Universidade de Oxford, é um evento com participação por convite que encoraja os alunos a desenvolverem suas habilidades de **Pensamento Computacional** por meio do uso de algoritmos e programas para resolver problemas (UNIVERSITY OF OXFORD, [s.d.]b).

O programa de graduação em computação do **Imperial College London** enfatiza princípios fundamentais, pensamento lógico e considerações de engenharia no design de sistemas (IMPERIAL COLLEGE LONDON, [s.d.]). Os alunos participam de aulas de laboratório e resolução de problemas, bem como de trabalhos de projeto e design. Os módulos centrais incluem as disciplinas: Matemática Discreta (7,5 ECTS); Matemática, Lógica e Raciocínio (7,5 ECTS); e Grafos e Algoritmos (7,5 ECTS), com módulos opcionais como Raciocínio Simbólico (6 ECTS) e Técnicas Computacionais (6 ECTS) (IMPERIAL COLLEGE LONDON, [s.d.]). Ainda que não tenha sido encontrada a expressão **Pensamento Computacional** explicitamente, os componentes da metodologia **DAAD** estão claramente integrados no currículo, com ênfase em abstração, lógica, algoritmos e representação de dados (IMPERIAL COLLEGE LONDON, [s.d.]).

A **Cardiff University** oferece uma disciplina dedicada ao **Pensamento Computacional** (CM1101, 20 créditos UK) no primeiro ano de seu curso de Ciência da Computação, juntamente com a disciplina de Resolução de Problemas com Python (10 créditos UK) e outra de Programação Orientada a Objetos com Java (20 créditos UK) (CARDIFF UNIVERSITY, [s.d.]). Isso indica um foco precoce dedicado ao **Pensamento Computacional**, porém espalhado por diversas disciplinas do currículo do curso.

## 3.3 Universidades na Europa Continental

Desde 2014, muitos países europeus têm revisado seus currículos de educação obrigatória para introduzir conceitos básicos de Ciência da Computação, preparando o terreno para o desenvolvimento de habilidades de **Pensamento Computacional** (EUROPEAN COMMISSION, 2021). A Comissão Europeia, por meio do seu Plano de Ação para a Educação Digital 2021-2027, reforça a importância da educação em computação como uma prioridade para aprimorar as habilidades e competências digitais (EUROPEAN COMMISSION, 2021).

A **ETH Zurich** (Suíça) enfatiza que o **Pensamento Computacional** vai além da programação e envolve as capacidades de abstração (ETH ZÜRICH, [s.d.]a). Para o ensino exclusivo de **Pensamento Computacional** a universidade adota um paradigma de sala de aula invertida, *flipped classroom* (BERGMANn, 2012), com vídeos e leituras para autoestudo, disponíveis online para seus alunos. O programa de Ciência da Computação também inclui, no primeiro ano, as disciplinas: Introdução a Programação (8 **ECTS**); Estruturas de Dados e Algoritmos (8 **ECTS**); e Projeto Digital e Arquitetura de Computador (6 **ECTS**) (ETH ZURICH, 2024). O modelo de sala de aula invertida da ETH Zurich (ETH ZÜRICH, [s.d.]a) demonstra **uma estratégia pedagógica que desloca a instrução direta para fora da sala de aula para permitir mais tempo para a resolução interativa de problemas durante as aulas**. A declaração explícita de que "o **Pensamento Computacional** é mais do que programar um computador, significa pensar em abstrações" (ETH ZÜRICH, 2020) indica um foco conceitual profundo. Sugerindo foco na aplicação de conteúdos normalmente integrados as disciplinas dos cursos de Ciência e Engenharia da Computação.

Na **Edinburgh University**, o programa de Ciência da Computação foca na compreensão, design, implementação e uso de sistemas computacionais, com conceitos centrais derivados da matemática, lógica e engenharia (UNIVERSITY OF EDINBURGH, [s.d.]). As disciplinas do primeiro ano incluem Introdução a Computação (20 créditos UK), Programação de Computadores (20 créditos UK) e, principalmente, Como Resolver Problemas Usando Computadores (créditos não disponíveis) (UNIVERSITY OF EDINBURGH, [s.d.]). A universidade enfatiza o trabalho prático, incluindo a construção de sistemas computacionais e trabalho experimental (UNIVERSITY OF EDINBURGH, [s.d.]).

A **École hôtelière de Lausanne (EHL)**, renomada instituição suíça de ensino superior, integra um módulo específico de "Pensamento Computacional" (88 horas: 30 horas de contato + 58 horas de estudo independente) em seu **programa de Bacharelado em Administração Hoteleira (EHL HOSPITALITY BUSINESS SCHOOL, 2024)**. Esta disciplina, oferecida no segundo semestre, representa 3.5 créditos **ECTS** e uma carga horária total de 88 horas, sendo 30 horas de contato direto com o professor e 58 horas destinadas ao estudo independente (EHL HOSPITALITY BUSINESS SCHOOL, 2024). **A inclusão de Pensamento Computacional em um programa de Administração Hoteleira demonstra o reconhecimento crescente de que essas competências transcendem as áreas tradicionais de ciência da computação**, aplicando-se a diversos campos profissionais (EHL HOSPITALITY BUSINESS SCHOOL, 2024). A oferta no segundo semestre indica seu papel como base fundamental para o desenvolvimento de habilidades de resolução estruturada de problemas, análise de dados e desenvolvimento de soluções inovadoras no setor hoteleiro. Esta abordagem da EHL alinha-se com as tendências globais de integração explícita do **Pensamento Computacional** nos currículos de graduação, independentemente da área de formação principal (WING, 2006).

## 3.4 Universidades na China

As universidades chinesas demonstram um compromisso significativo, porém diversificado, com o Pensamento Computacional, adotando modelos distintos de implementação:

A **Universidade de Shandong** adota um centralizado, oferecendo a disciplina Introdução à Computação e Design de Programas (4,5 créditos) como disciplina obrigatória em seus cursos de Ciência e Tecnologia da Computação, Inteligência Artificial e Ciência de Dados (UNIVERSIDADE DE SHANDONG, 2020). Embora divulgado internacionalmente como a disciplina Pensamento Computacional e Programação (5 créditos) (UNIVERSIDADE DE SHANDONG, [s.d.]a), o currículo oficial visa "melhorar sistematicamente a capacidade de resolução de problemas dos alunos, a capacidade de sistemas de computador, o pensamento inovador e a capacidade de inovação" (UNIVERSIDADE DE SHANDONG, [s.d.]b), alinhando-se diretamente com os princípios do **Pensamento Computacional** através de uma abordagem robusta e intensiva de 88 horas totais (UNIVERSIDADE DE SHANDONG, 2020).

A **Universidade de Pequim (PKU)**, a PKU implementa um **Modelo de Literacia Distribuída**, onde o **Pensamento Computacional** é integrado em disciplinas aplicadas oferecidas por diversos departamentos. Um exemplo notável é a disciplina Fundamentos da Linguagem Python e Aplicações (3,5 créditos) oferecida pela **Escola de Ciências da Terra e do Espaço**, que declara explicitamente o objetivo de "guiar os alunos a construir ativamente um modelo de pensamento computacional" e "resolver problemas através de algoritmos de programa para aprofundar a sua compreensão da linguagem de programação" (UNIVERSIDADE DE PEQUIM, [s.d.]a). Adicionalmente, existe a disciplina Pensamento Computacional (3 créditos) (UNIVERSIDADE DE PEQUIM, [s.d.]b), confirmando uma estratégia de promover o **Pensamento Computacional** como competência transdisciplinar.

A **Universidade de Tsinghua** representa o modelo interdisciplinar integrado considerando o **Pensamento Computacional** uma competência intrínseca integrada ao currículo principal (UNIVERSIDADE DE TSINGHUA, 2022). A universidade é pioneira na criação de programas híbridos como Engenharia Química e Inteligência Artificial no Tanwei College, onde o **Pensamento Computacional** está incorporado na própria identidade do programa, visando formar engenheiros químicos que sejam capazes de "desconstruir a lógica algorítmica e aplicá-la para resolver problemas químicos" (UNIVERSIDADE DE TSINGHUA, 2025). Globalmente, Tsinghua oferece a "Especialização em Estruturas de Dados e Algoritmos" na plataforma Coursera (COURSERA, 2024), listando explicitamente "Pensamento Computacional" como competência-chave (UNIVERSIDADE DE TSINGHUA, [s.d.]).

Esta análise revela três modelos distintos de implementação estratégica: o modelo centralizado e pragmático de Shandong garante conformidade eficiente com diretrizes nacionais; o modelo distribuído de Pequim promove a criação de alunos com capacidade de **Pensamento Computacional** em toda a universidade; e o modelo integrado de Tsinghua representa a vanguarda na fusão fundamental entre disciplinas. A diversidade destes modelos parece indicar um ecossistema educacional dinâmico onde as universidades exercem autonomia significativa para inovar pedagogicamente dentro do quadro de políticas nacionais chinesas para educação em Inteligência Artificial e competências digitais (CHINA, 2017), revelando não um sistema monolítico, mas um ambiente de experimentação e competição entre as principais instituições do país.

A Table 3.5 oferece uma visão estruturada e comparativa de como universidades líderes em diferentes regiões abordam a educação em Pensamento Computacional, com foco nos elementos **DAAD**. Esta análise permite identificar padrões comuns e pontos fortes únicos, fornecendo informações valiosas para o design de um currículo de 80 horas, ao apresentar diversos modelos de integração e ênfase pedagógica.

Table 3.5: Exemplos de Universidades e a Integração de **Pensamento Computacional/DAAD** em seus Currículos de Graduação.

| Universidade | País/Região | Tipo de Curso/Integração | Disciplinas Principais | Créditos/Carga Horária | Elementos DAAD/PC Enfatizados | Linguagens/Ferramentas | Destaques Pedagógicos |
|---|---|---|---|---|---|---|---|
| **Carnegie Mellon University** | EUA | Centro dedicado de PC, Currículo específico | Princípios da Computação; Estrutura de Dados e Algoritmos; Codificação e PC com VEX V5 | 10 créditos; 12 créditos; curso de extensão | Decomposição, Abstração, Algoritmização, Depuração | VEX V5 Robotics | PROBEs (explorações orientadas a problemas), aplicação em contextos do mundo real |
| **MIT** | EUA | Requisito mínimo para todos os estudantes | Introdução ao PC em Python (6.100A); PC e Ciência de Dados (6.100B) | 12 créditos cada | Raciocínio rigoroso, comunicação, impacto transformador | Python | Desenvolvimento de confiança para programas úteis independente da área de estudo |
| **Penn State University** | EUA | Integrado no currículo de Ciência da Computação | Disciplinas do curso de CS | Não especificado | Conceituação e implementação, abstração em múltiplos níveis, análise de código | Não especificado | Ênfase em comportamento, eficiência e correção do código |
| **University of Wisconsin-Madison** | EUA | Disciplina específica | Resolução de Problemas Usando Computadores (COMP SCI 310) | 3 créditos | Abstração, decomposição de problemas | Linguagens de manipulação simbólica, pacotes de software | Métodos computacionais para resolução de problemas |

| Universidade | País/Região | Tipo de Curso/Integração | Disciplinas Principais | Créditos/Carga Horária | Elementos DAAD/PC Enfatizados | Linguagens/Ferramentas | Destaques Pedagógicos |
|---|---|---|---|---|---|---|---|
| **University of Texas at Austin** | EUA | Disciplina introdutória | Programação Introdutória | Não disponível | Desenvolvimento de algoritmos, decomposição, depuração | Não especificado | Conceitos fundamentais de Ciência da Computação |
| **Cambridge University** | Reino Unido | Instituto Cambridge Mathematics, currículo integrado | Algoritmos 1; Lógica e Algoritmos; Computational Thinking Challenge | Curso geral | Abstração, lógica, algoritmos, representação de dados | Não especificado | Relação profunda entre PC e pensamento funcional, projeto de pesquisa e avaliação digital |
| **Oxford University** | Reino Unido | Integrado em múltiplos cursos | Cursos de CS, Matemática e CS, CS e Filosofia | Não especificado | Design de programas, conceitos de alto nível | Não especificado | UK Bebras Challenge, Oxford University Computing Challenge, ênfase em leitura de materiais conceituais |
| **Imperial College London** | Reino Unido | Programa de graduação em computação | Matemática Discreta; Matemática, Lógica e Raciocínio; Grafos e Algoritmos | 7,5 ECTS cada (centrais); 6 ECTS (opcionais) | Princípios fundamentais, pensamento lógico, algoritmos | Não especificado | Laboratórios, resolução de problemas, projetos de design |

| Universidade | País/Região | Tipo de Curso/Integração | Disciplinas Principais | Créditos/Carga Horária | Elementos DAAD/PC Enfatizados | Linguagens/Ferramentas | Destaques Pedagógicos |
|---|---|---|---|---|---|---|---|
| **Cardiff University** | Reino Unido | Disciplina dedicada no 1º ano | Pensamento Computacional (CM1101); Resolução de Problemas com Python; POO com Java | 20 créditos UK; 10 créditos UK; 20 créditos UK | PC dedicado, resolução de problemas | Python, Java | Foco precoce distribuído por diversas disciplinas |
| **ETH Zurich** | Suíça | Paradigma de sala de aula invertida | Introdução a Programação; Estruturas de Dados e Algoritmos; Projeto Digital | 8 ECTS; 8 ECTS; 6 ECTS | Abstração ("PC é mais que programar") | Não especificado | Flipped classroom, vídeos e leituras online, autoestudo |
| **Edinburgh University** | Escócia | Programa de Ciência da Computação | Introdução a Computação; Programação de Computadores; Como Resolver Problemas | 20 créditos UK cada | Compreensão, design, implementação de sistemas | Não especificado | Trabalho prático, construção de sistemas, trabalho experimental |
| **École Hôtelière de Lausanne (EHL)** | Suíça | Módulo específico em Administração Hoteleira | Pensamento Computacional | 3,5 ECTS (88h: 30h contato + 58h estudo) | Base para resolução estruturada, análise de dados | Não especificado | Aplicação transdisciplinar, integração em área não-tecnológica |

| Universidade | País/Região | Tipo de Curso/Integração | Disciplinas Principais | Créditos/Carga Horária | Elementos DAAD/PC Enfatizados | Linguagens/Ferramentas | Destaques Pedagógicos |
|---|---|---|---|---|---|---|---|
| **Universidade de Shandong** | China | Modelo centralizado, disciplina obrigatória | Introdução à Computação e Design de Programas / PC e Programação | 4,5 créditos / 5 créditos (88h totais) | Resolução de problemas, capacidade de sistemas, inovação | Não especificado | Foco em conformidade com diretrizes nacionais, abordagem robusta |
| **Universidade de Pequim (PKU)** | China | Modelo de Literacia Distribuída | Fundamentos Python e Aplicações; Pensamento Computacional | 3,5 créditos; 3 créditos | Construção ativa de modelo de PC, algoritmos | Python | Integração em disciplinas aplicadas por diversos departamentos |
| **Universidade de Tsinghua** | China | Modelo interdisciplinar integrado | Engenharia Química e IA; Especialização em Estruturas de Dados | Programas híbridos | Desconstrução lógica algorítmica, aplicação interdisciplinar | Não especificado | Programas híbridos inovadores, competência intrínseca integrada, ofertas globais (Coursera) |

# 4 Definição e Princípios da Metodologia DAAD

O **Pensamento Computacional** é uma estrutura que descreve um conjunto de habilidades de pensamento crítico e resolução de problemas, que tem ganhado significativa relevância como uma forma eficaz de ensinar essas habilidades em ambientes educacionais formais (BRACKMANN, 2017; FOREDUCATION EDTECH, 2023). Embora não seja a única abordagem para desenvolver essas competências, o **Pensamento Computacional** oferece uma maneira de analisar problemas para gerar soluções automatizadas ou semi-automatizadas que utilizam as capacidades únicas das tecnologias computacionais (VALENTE, 2016).

Jeannette Wing define o **Pensamento Computacional** como *"os processos de pensamento envolvidos na formulação de problemas e suas soluções de forma que as soluções possam ser efetivamente executadas por um agente de processamento de informações"* (WING, 2006, p. 33). Ela também o descreve como a resolução de problemas, o design de sistemas e a compreensão do comportamento humano, baseando-se em conceitos estruturantes da ciência da computação (WING, 2006; TEDRE & DENNING, 2019). O **Pensamento Computacional** é uma combinação de hábitos mentais disciplinados, atitudes de perseverança e habilidades interpessoais essenciais (CSTA; ISTE, 2011). Ele se distingue de outras abordagens de resolução de problemas, como o pensamento científico ou o *design thinking*, principalmente por seu foco em soluções algorítmicas e no desenvolvimento de sistemas que envolvem processamento de informações (DORST; CROSS, 2021).

De acordo com Wing (2006), o Pensamento Computacional estrutura-se em componentes fundamentais que transcendem a mera programação: a **decomposição** (fragmentação de problemas complexos em partes gerenciáveis), o **reconhecimento de padrões** (identificação de similaridades e tendências em diferentes contextos), a **abstração** (seleção de elementos essenciais e eliminação de detalhes irrelevantes) e o **design algorítmico** (criação de instruções passo a passo para solução automatizada), complementados por processos de **generalização** (aplicação de soluções a problemas análogos) e **modelagem** (representação de sistemas do mundo real através de estruturas computacionais), formando uma abordagem holística para a resolução eficiente de problemas (WING, 2006, p. 34-35).

A eficácia do **Pensamento Computacional** parece advir da maneira como essas habilidades se complementam e se reforçam mutuamente, em um processo iterativo que vai da compreensão do problema à sua solução e refinamento. Ao invés de serem habilidades isoladas, elas funcionam em sinergia, permitindo que indivíduos abordem a complexidade de forma sistemática.

A metodologia **DAAD**, acrônimo de **P**roblem Decomposition, **A**bstraction, Algorithm **D**esign e **D**ebugging, estrutura-se como uma aplicação pedagógica direta dos componentes fundamentais do Pensamento Computacional definidos por Wing (2006) adaptados para a realidade dos cursos de graduação no Brasil especificamente aqueles de ciências, matemática, computação e engenharia. A fase **Problem Decomposition**, decomposição do problema, é a manifestação prática da **decomposição**, fragmentando desafios complexos em subproblemas gerenciáveis e facilmente descritíveis em estruturas de comandos ou fluxogramas. A fase **Abstraction** reflete diretamente o pilar homônimo do modelo **Pensamento Computacional**, filtrando detalhes irrelevantes para isolar padrões essenciais, incorporando implicitamente o *reconhecimento de padrões*. A fase **Algorithm Design** materializa o pilar _design algorítmico*_ do trabalho de Wind (2006), transformando abstrações em sequências executáveis,

por meio de ferramentas de apoio ao **Raciocínio Algorítmico** como fluxogramas e pseudocódigos, enquanto promove a **generalização** ao transferir soluções para contextos análogos, porém mais amplos. Finalmente, o **Debugging** substitui o processo de **modelagem** computacional através de refinamento iterativo, validando soluções contra comportamentos esperados do sistema (WING, 2006). O **Debugging** entra como ferramenta de depuração. Não apenas no sentido computacional de encontrar erros em programas. Aqui **Debugging** é usada por sua ligação com o domínio da computação. Porém em um sentido mais amplo, no sentido de depurar, de purificar substâncias, de corrigir erros e de eliminar elementos indesejados. Assim, ainda que esta seja uma fase própria do processo de aprendizado, o **Debugging** representa o processo de correção e adaptação que será empregado em todos o processos de ensino da Metodologia **DAAD**. Esta sinergia posiciona o **DAAD** como uma estrutura operacional para consolidar a teoria do **Pensamento Computacional** em práticas educacionais tangíveis, mantendo a essência holística da proposta original.

O **Pensamento Computacional** e seus componentes transcendem a ciência da computação, constituindo-se como **habilidades transferíveis** aplicáveis em múltiplos domínios do conhecimento (WING, 2006)(SBC, 2017). Essa natureza multifacetada confere ao Pensamento Computacional um caráter **metacognitivo**, capacitando os indivíduos a refletirem sobre seus próprios processos mentais e a abordarem problemas complexos de forma sistemática em qualquer contexto (SBC, 2017). Conforme destacado por Wing (2006), a essência dessa competência reside na capacidade de "reformular problemas aparentemente intratáveis em formas solucionáveis" (WING, 2006), habilidade que se manifesta, por exemplo, quando estudantes de biologia decompõem ecossistemas em modelos algorítmicos ou quando artistas utilizam abstração para criar padrões visuais computacionais (BRACKMANN, 2017)(RYCRAFT-SMITH & CONNOLLY, 2019).

A aplicação sistemática do Pensamento Computacional cultiva uma **mentalidade analítica e inovadora**, beneficiando áreas tão diversas quanto:
- **Humanidades**: análise crítica de dados históricos por decomposição temporal (SBC, 2017);
- **Artes**: geração de composições musicais através de reconhecimento de padrões (BRACKMANN, 2017);
- **Ciências Sociais**: modelagem de dinâmicas populacionais via algoritmos (RYCRAFT-SMITH & CONNOLLY, 2019).

Sua relevância como **competência universal do século XXI** é evidenciada pela integração curricular transversal. Países como o Brasil já incorporam seus pilares, decomposição, padrões, abstração, algoritmos, em diretrizes para educação básica (SBC, 2017), enquanto universidades europeias o inserem em cursos de filosofia e economia (RYCRAFT-SMITH & CONNOLLY, 2019).

## 4.1 Metodologias Semelhantes ao DAAD

A abordagem dos **Quatro Pilares do Pensamento Computacional**, desenvolvida pela University of York, estrutura-se em decomposição, reconhecimento de padrões, abstração e pensamento algorítmico. Essa metodologia enfatiza a fragmentação de problemas complexos e a identificação de regularidades, mas não formaliza etapas de validação pós-implementação, limitando-se à concepção teórica de soluções (UNIVERSITY OF YORK, 2020) (SENTANCE, S. et al, 2017). Já a **Abordagem por Atividades Desplugadas**, proposta por Brackmann (2017), prioriza intervenções pedagógicas sem uso de tecnologia, utilizando recursos físicos, como quebra-cabeças, para desenvolver abstração e decomposição. Embora eficaz em contextos com infraestrutura limitada, sua aplicação tende a focar em modelagem de *hardware* em detrimento do design sistemático de algoritmos, além de carecer de sequencialidade didática clara (BRACKMANN, 2017, p. 42-45) (BRACKMANN, 2017)(SENTANCE, S. et al, 2017).

O ***framework* de Wing (2006)**, base teórica seminal, define pilares como decomposição, abstração, design algorítmico e generalização. Wing enfatiza que a generalização de soluções permite aplicação em múltiplos contextos, porém não integra explicitamente mecanismos de depuração (*debugging*) ou validação iterativa, concentrando-se na fase de concepção em vez do refinamento prático (WING, 2006, p. 34) (SENTANCE, S. et al, 2017)(KURKOVSKY, 2013). Essas lacunas são supridas pelo **DAAD**, que incorpora o *Debugging* como etapa crítica para testar robustez, eficiência e escalabilidade de algoritmos em cenários reais – como em sistemas embarcados ou otimização NP-difícil –, consolidando um ciclo completo de desenvolvimento (KURKOVSKY, 2013; SENTANCE et al., 2017) (SENTANCE, S. et al, 2017). A Table 4.1 resume estas metodologias.

Table 4.1: Propostas metodológicas de aplicação pedagógica dos conceitos de **Pensamento Computacional**

| Metodologia | Instituição/Referência | Componentes-Chave | Diferenças para o DAAD |
|---|---|---|---|
| **Quatro Pilares do PC** | University of York | 1. Decomposição2. Reconhecimento de padrões3. Abstração4. Pensamento algorítmico | • Substitui *Debugging* por *Reconhecimento de padrões* • Não inclui refinamento iterativo após implementação |
| **Abordagem por Atividades Desplugadas** | Brackmann (2017) | • Problemas físicos (ex.: quebra-cabeças) • Modelagem conceitual sem código • Ênfase em abstração | • Foco em *hardware* vs. *solução algorítmica* • Menos estruturada em etapas sequenciais |
| ***framework* de Wing (2006)** | Wing (2006) | • Decomposição • Abstração • Design algorítmico • Generalização | • Não formaliza *Debugging* como pilar independente • *Generalização Validação prática iterativa* |

## 4.2 Inovações do DAAD

A diferença *sine qua non* do **DAAD** em relação aos outros modelos é a integração explícita de **Debugging***. Uma fase de aprendizado dedicada a depuração, que não apenas identifica erros, mas também valida soluções em contextos reais. Essa abordagem corrige a lacuna de modelos que ignoram o refinamento pós-implementação, promovendo uma compreensão mais profunda e prática do desenvolvimento de algoritmos. O Debugging adiciona um pilar dedicado à validação iterativa, ausente nos *framework_s tradicionais corrigindo a lacuna de modelos que ignoram o refinamento pós-implementação das soluções encontradas. Na Abordagem de York (2020), o reconhecimento de padrões é estático, identificação de similaridades, enquanto no **DAAD** a abstração é dinâmica, preparação para depuração. O modelo original de Wing (2006) trata generalização como resultado, enquanto no **DAAD** o Debugging torna-a processual através de iteração. Finalmente, a Abordagem -unplugged* de Brackmann (2017) enfatiza modelagem física, mantém abstração como conceito teórico, já no DAAD vincula-se à rastreabilidade durante validação, enquanto o **DAAD** prioriza soluções algorítmicas e validação prática.

A estrutura do **DAAD** também inclui uma . Sequência pedagógica otimizada em uma ordenação lógica: *Problema → Abstração → Algoritmo → Validação.* O que contrasta com os modelos fragmentados (ex.: York, Wing) que não articulam nenhuma ordem de transição entre etapas.

Além disso, como o foco é em engenharia de software aplicada a Ciência e Engenharia da Computação, o **DAAD** enfatiza a criação de algoritmos robustos e eficientes, alinhando-

se às demandas do mercado de trabalho. Isso inclui, a aplicação de técnicas avançadas de depuração, como testes de estresse e análise de *edge cases*, casos limites, que vão além da simples correção de erros sintáticos. Suportando processos de depuração sistemática alinhados as demandas industriais.

A relação entre as fases do **DAAD** e a solução de problemas pode ser vista por meio da análise de problemas símples típicos das disciplinas introdutórias dos cursos de Ciência e Engenharia da Computação:

1. **Gerenciamento de Tarefas Acadêmicas**: usando o trabalho de Kerzner (2017) como base para gestão de projetos acadêmicos, teremos:

- **Problem Decomposition**: dividir planejamento semestral em: disciplinas, tarefas semanais, prazos de entrega.
- **Abstraction**: definir modelo mínimo (prioridade, tempo estimado, prazo) ignorando detalhes não-essenciais.

- **Algorithm Design**: criar algoritmo de priorização baseado em: `(dias_restantes / complexidade) * peso_nota`.

- **Debugging**: simular sobrecarga (ex.: 3 trabalhos no mesmo dia) e validar ajustes automáticos

2. **Organização de Coleções Digitais**: usando como base o trabalho de Rowley (2007) sobre organização de informações digitais, teremos:

- **Problem Decomposition**: fragmentar catálogo de mídias (músicas/livros) por: gênero, autor, ano.

- **Abstraction**: representar itens apenas com metadados essenciais (título, autor, ano).

- **Algorithm Design**: desenvolver sistema de busca por similaridade (ex.: "encontrar livros como X").

- **Debugging**: testar com entradas ambíguas (ex.: títulos parecidos, autores homônimos).

3. **Otimização de Rotas no Campus**: usando o trabalho de Even (2011) sobre algoritmos de grafos aplicados a problemas cotidianos, teremos:

- **Problem Decomposition**: dividir mapa do campus em setores e conexões entre prédios.

- **Abstraction**: modelar caminhos como grafos simples (nós = prédios, arestas = caminhos).

- **Algorithm Design**: implementar algoritmo para menor caminho considerando: distância, escadas, acesso para deficientes.

- **Debugging**: validar com rotas bloqueadas (ex.: obras) e pontos de interesse

4. **Otimização de Busca em Conjuntos de Dados**: adaptação de princípios de divisão e conquista para problemas de busca (SEDGEWICK, 2011).

- **Problem Decomposition**: dividir grandes conjuntos de dados (ex.: 10.000 registros) em subconjuntos menores por faixas de valores (ex.: intervalos numéricos ou categorias alfabéticas).

- **Abstraction**: modelar registros apenas com atributos essenciais para busca (ex.: ID numérico, chave primária), ignorando metadados irrelevantes.
- **Algorithm Design**: projetar uma estratégia híbrida de busca binária em subconjuntos ordenados com busca sequencial em partições pequenas.

- **Debugging**: validar eficiência com conjuntos desbalanceados (ex.: 90% dos registros concentrados em 10% das partições) usando perfis de desempenho.

A metodologia **DAAD** foi concebida ir além das disciplinas introdutórias e além dos cursos de Ciência e Engenharia da Computação. O objetivo é que o **DAAD** ajude a criar a capacidade de resolver problemas complexos de forma sistemática, promovendo uma compreensão profunda dos processos envolvidos na criação de soluções em qualquer área do conhecimento. Entretanto, como a prova de conceito aqui proposta diz respeito aos cursos de Ciência e Engenharia da Computação, a seguir são apresentados exemplos de aplicação do **DAAD** em contextos mais avançados, que demonstram sua versatilidade e aplicabilidade em problemas complexos e multidisciplinares.

1. **Otimização Algorítmica para Problemas NP-Difíceis**: usando um modelo de decomposição baseado em Cook (2011) para problemas NP-completos, com métricas de validação de Dolan-Moré (2002).

- **Problem Decomposition**: fragmentação de problemas de otimização combinatorial (ex.: *Traveling Salesman Problem*) em subproblemas de roteamento local e conexões inter-regionais, permitindo abordagens *divide-et-conquer*, dividir e conquistar.

- **Algorithm Design**: implementação de meta-heurísticas (ex.: *Simulated Annealing* com resfriamento adaptativo) para busca de soluções -ótimas em espaços de estado complexos.

- **Debugging**: validação via *gap analysis* entre soluções heurísticas e limites teóricos (ex.: comparação com *Held-Karp bound*), utilizando perfis de desempenho (*performance profiles*) para avaliação estatística .

2. **Engenharia de Sistemas Embarcados Críticos**: usando ramework de abstração conforme definido por Wilhelm (2008) para sistemas tempo-real, com validação por Henzinger (2007), teremos:

- **Abstraction**: modelagem de invariantes temporais e de consumo energético através de **W**orst-**C**ase **E**xecution **T**ime, **WCET**, e *power-state machines*, filtrando variáveis não essenciais ao cumprimento de restrições rígidas de tempo real.

- **Algorithm Design**: síntese de algoritmos com garantias formais (ex.: controle PID com provas de estabilidade Lyapunov) atendendo a requisitos de segurança ISO 26262 ASIL-D.

- **Debugging**: verificação **H**ardware-**I**n-the-**L**oop*, **HIL**, com injeção de falhas sistêmicas e análise de violações de* deadlines* via *tracing* em tempo real.

3. **Desenvolvimento de Pipeline de Aprendizado de Máquina**: usando com referência o trabalho de **GÉRON (2019)**, que descreve a construção de sistemas de aprendizado de máquina como um processo iterativo e sistemático, teremos:

- **Problem Decomposition**: fragmentação do fluxo de trabalho em coleta de dados, pré-processamento, seleção de modelo, treinamento e avaliação.

- **Abstraction**: identificação de características essenciais nos dados (ex.: seleção de *features* via PCA ou análise de importância) .

- **Algorithm Design**: projeto de arquitetura de redes neurais ou ajuste de hiperparâmetros de algoritmos de classificação.

- **Debugging**: validação cruzada, análise de *overfitting* e ajuste de modelos com base em métricas de precisão/recall.

4. **Desenvolvimento de Sistemas Concorrentes**: usando como referência o trabalho de **HERLIHY (2015)**, que aborda a construção de sistemas concorrentes e distribuídos, teremos:

- **Problem Decomposition**: divisão do sistema em tarefas concorrentes (ex.: *threads* para E/S, processamento e comunicação).

- **Abstraction**: isolamento de seções críticas e recursos compartilhados (ex.: buffer de mensagens).

- **Algorithm Design**: criação de protocolos de sincronização (ex.: semáforos, *mutexes*) e algoritmos de exclusão mútua.

- **Debugging**: teste de estresse para detecção de *deadlocks* e *race conditions* com ferramentas como *Valgrind* ou *TSan* .

5. **Desenvolvimento de Sistemas Distribuídos**: este caso requer a aplicação dos princípios de decomposição hierárquica e depuração em falhas não determinísticas, seguindo o modelo de **Tanenbaum (2015)** para sistemas distribuídos .

- **Problem Decomposition**: divisão de sistemas complexos em microsserviços independentes (ex.: separação de módulos de autenticação, processamento de pagamentos e gerenciamento de dados) para tratamento paralelo.
- **Algorithm Design**: implementação de protocolos de consenso como *Raft* ou *Paxos* para garantir consistência entre nós distribuídos (ONGARO, 2014).

- **Debugging**: simulação de falhas em cascata e testes de partição de rede com ferramentas como *Chaos Monkey* (NETFLIX,2025) para validar resiliência .

6. **Ciência de Dados em Bioinformática**: adaptação do **framework Bioconductor** para validação iterativa em análise genômica, conforme **Huber (2015)**

- **Abstraction**: identificação de variáveis genéticas essenciais em sequências de DNA (ex.: polimorfismos de nucleotídeo único - SNPs) ignorando "ruído" biológico.
  **Algorithm Design**: Criação de pipelines de alinhamento usando **técnicas de seleção adaptativa** (ex.: integração BLAST/MinKNOW para enriquecimento de espécies raras) (PAYNE, 2022)(LOMAN, 2015).

- **Debugging**: validação estatística de falsos positivos/negativos através de reamostragem *bootstrap* e métricas de precisão.

### 4.2.1 Validação do DAAD com Base em Evidências

Estudos mostram que metodologias com **4+ etapas estruturadas** aumentam em 40% a retenção de conceitos de PC entre estudantes de computação . O *Debugging*, quando integrado desde o projeto inicial, reduz erros de implementação em 62% comparado a abordagens reativas .

## 4.3 Decomposição: Quebrando a Complexidade

A **D**ecomposição é o processo analítico pelo qual problemas ou conceitos complexos são divididos em partes menores e mais gerenciáveis (EDTECHBOOKS, [s.d.]). Essa abordagem é necessária para a resolução eficaz de problemas, pois transforma situações que inicialmente parecem esmagadoras em elementos mais simples e acessíveis (KIM; LEE, 2018 apud KIM, 2021). Ao dividir um problema grande em subproblemas, torna-se mais fácil analisar cada componente isoladamente e, posteriormente, integrar as soluções para resolver o problema original (VOOGT et al., 2015 apud KIM, 2021).

No contexto da engenharia, a decomposição é empregada para desmembrar tarefas complexas, como a simulação de construção, em segmentos menores e mais fáceis de gerenciar (EDTECHBOOKS, [s.d.]). Na ciência da computação, a decomposição visa quebrar um problema ou sistema complexo em partes que são mais fáceis de conceber, entender, programar e manter (Guzdial, 2015 apud KIM, 2021). Exemplos clássicos incluem a divisão da tarefa de fazer um bolo em etapas menores, como preparar a massa, assar o bolo, fazer a cobertura e aplicá-la (BARES et al., 2019 apud KIM, 2021).

## 4.4 Abstração: Focando no Essencial

A **A**bstração é uma técnica central do **Pensamento Computacional** que se concentra em identificar informações importantes e relevantes, enquanto ignora detalhes desnecessários ou irrelevantes, facilitando uma compreensão mais clara das questões essenciais (EDTECHBOOKS, [s.d.]) referentes ao problema que precisa ser resolvido. A abstração é o processo de generalizar detalhes concretos para direcionar a atenção para aspectos de maior importância (KIM; LEE, 2018 apud KIM, 2021).

A abstração é considerada o processo de pensamento de mais alto nível no **Pensamento Computacional**, conferindo a capacidade de escalar e gerenciar a complexidade (CAMBRIDGE, [s.d.]). A sua presença é tão difundida na ciência da computação que a sua descrição concisa é um desafio, e embora haja um consenso sobre a sua centralidade, as definições exatas podem variar entre os investigadores (PERKINS, 2012 apud KIM, 2021). Exemplos ilustrativos de abstração incluem o uso de mapas, que simplificam o mundo real ao omitir detalhes desnecessários, ou a forma como os sistemas computacionais ocultam as suas complexidades internas do utilizador em interfaces gráficas, apresentando apenas a informação relevante para a interação (*unplugged*, [s.d.]).

## 4.5 Algoritmização: Desenvolvendo Soluções Sistemáticas

A **A**lgoritmização, ou Design de Algoritmos, envolve a criação de instruções passo a passo ou procedimentos para resolver um problema (EDTECHBOOKS, [s.d.]). Um algoritmo é definido como uma sequência finita de instruções matematicamente rigorosas, tipicamente

utilizadas para resolver uma classe específica de problemas ou para realizar uma computação (KIM; LEE, 2018 apud KIM, 2021).

O pensamento algorítmico baseia-se na premissa de que as soluções para os problemas não se limitam a respostas pontuais, mas sim a algoritmos que podem fornecer respostas sempre que necessário para casos gerais (CAMBRIDGE, [s.d.]). É o processo de construir um esquema de passos ordenados que podem ser seguidos para fornecer soluções para todos os problemas constituintes necessários para resolver o problema original (*unplugged*, [s.d.]). A capacidade de expressar uma solução na forma de um algoritmo demonstra uma compreensão mais profunda do problema (CAMBRIDGE, [s.d.]). A eficiência, em termos de velocidade ou uso de memória, é uma consideração importante na criação de algoritmos (CAMBRIDGE, [s.d.]) sempre que o objetivo final for transformar o algoritmo em uma solução computacional. Os algoritmos podem ser expressos de várias formas, incluindo linguagem natural, pseudocódigo, fluxogramas e diagramas (KIM; LEE, 2018 apud KIM, 2021).

## 4.6 Depuração: Identificando e Corrigindo Erros

O **D**epuração, ou *debugging* é o processo de encontrar e corrigir erros, ou *bugs*, no código-fonte de qualquer software, algoritmo ou sistema (EDTECHBOOKS, [s.d.]). É uma etapa indispensável para garantir que uma solução encontrada funcione conforme o esperado (*unplugged*, [s.d.]).

O processo de depuração envolve a identificação do erro, a análise de sua causa, registrando as mudanças de estado do programa e os valores dos dados, a correção do problema e a validação da correção por meio de testes (LI; CHENG, 2019 apud KIM, 2021). É uma atividade iterativa que contribui para a construção do conhecimento e o aprendizado de estratégias de resolução de problemas (*unplugged*, [s.d.]). A depuração permite que os programadores localizem e resolvam bugs, melhorando a qualidade do software e a experiência do usuário final (LI; CHENG, 2019 apud KIM, 2021).

## 4.7 O Papel da DAAD na Educação em Ciência e Engenharia da Computação

O **Pensamento Computacional**, incluindo os elementos da metodologia **DAAD**, é considerado uma habilidade básica e estruturante para o desenvolvimento científico, tecnológico e econômico no século XXI (Review on the teaching..., [s.d.]). Sua relevância se estende além das fronteiras da ciência da computação, sendo essencial para a resolução de problemas em diversas áreas *STEM* (Ciência, Tecnologia, Engenharia e Matemática) e *No-STEM* (Review on the teaching..., [s.d.]). Profissionais da computação, por sua própria natureza, utilizam intrinsecamente essas habilidades: pensar abstratamente, operar em múltiplos níveis de abstração, abstrair ideias para gerenciar a complexidade, e empregar a iteração, a depuração e o teste de software como práticas rotineiras (Review on the teaching..., [s.d.]).

O aprendizado de técnicas que permitam o desenvolvimento de um **Pensamento Computacional** atua como um grampo P suportando o aluno na escalada para o desenvolvimento das habilidades necessárias a solução de problemas com o uso de máquinas. Além disso, estas habilidades são altamente transferíveis para outras áreas do conhecimento (Review on the teaching..., [s.d.]). Isso significa que as competências desenvolvidas não se restringem aos estudiosos da ciência da computação, mas possuem o potencial de irradiar para todo o campo da ciência e tecnologia.

Observa-se que o **Pensamento Computacional** é consistentemente descrito como uma "habilidade fundamental" (Review on the teaching..., [s.d.]), uma "abordagem generalizada para a resolução de problemas" (Review on the teaching..., [s.d.]) e até mesmo um "novo

paradigma que expande as habilidades cognitivas dos alunos" (Computational Thinking Exercises, [s.d.]). Essa caracterização parece ir além do simples ensino de um conjunto de ferramentas ou técnicas. Se o **Pensamento Computacional** é um "novo paradigma", isso implica uma transformação na forma como os alunos abordam qualquer problema, não apenas aqueles relacionados à codificação. O objetivo final não é apenas formar programadores proficientes, mas cultivar uma "mentalidade computacional" (A Pedagogical *framework*..., [s.d.]) que capacite os indivíduos a "pensar de forma analítica e crítica" (Benefits and Challenges..., 2021). Para que essa transformação cognitiva ocorra, o currículo não deve se limitar à sintaxe de programação, mas deve focar no desenvolvimento dos processos cognitivos subjacentes e complementares. Nesse contexto, as atividades *unplugged*, aquelas que não envolvem computadores, tornam-se particularmente valiosas (Computational Thinking School..., [s.d.]). Elas permitem que os alunos compreendam os aspectos conceituais da metodologia **DAAD** sem a carga cognitiva imediata da programação e do ambiente computacional, estabelecendo uma base sólida para futuras atividades e permitindo a transferência destas habilidades para as atividades *plugged*, atividades realizadas com máquinas, mais complexas.

# 5 Estratégias Pedagógicas e Materiais Didáticos para o Ensino de DAAD

O ensino eficaz dos componentes do **Pensamento Computacional** (Decomposição, Abstração, Algoritmização, Debugging) exige estratégias pedagógicas que transcendem as aulas expositivas tradicionais, promovendo o engajamento ativo e a aplicação prática. Uma abordagem progressiva e integrada entre atividades *unplugged*, desconectadas, e *plugged*, conectadas, é a base deste engajamento. **Iniciar com atividades *unplugged* permite que os alunos construam uma compreensão conceitual sólida sem a barreira da sintaxe de programação (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021)**. Posteriormente, essas compreensões conceituais são reforçadas e aplicadas por meio de desafios de programação *plugged* (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021). Essa progressão didática garante que os alunos compreendam os "porquês" antes de se aprofundarem nos "comos", otimizando o processo de aprendizado.

## 5.1 Abordagens Pedagógicas Eficazes

A **Metodologia Flipped Classroom**, aplicada pela ETH Zurich, é uma estratégia pedagógica que inverte a dinâmica tradicional de ensino. **Flipped Classroom**, é a expressão em inglês para sala de aula invertida e representa uma abordagem pedagógica que reverte a sequência do processo de ensino-aprendizagem. Tradicionalmente, a introdução a novos conceitos ocorre em sala de aula, seguida por atividades de reforço em casa. Nesta metodologia, o estudo inicial do conteúdo é realizado pelos alunos fora do ambiente de aula, por meio de materiais previamente fornecidos. Esse preparo prévio permite que o tempo em sala de aula seja otimizado para a prática, discussão e resolução de problemas, com o professor atuando como um orientador. A **Metodologia Flipped Classroom** pretende a promoção da autonomia do aluno, visto que o ritmo de assimilação do conteúdo inicial é adaptado às suas necessidades individuais. O foco das interações presenciais se desloca para a aplicação prática do conhecimento, o que pode fortalecer a compreensão e a retenção. O modelo permite que o professor ofereça um suporte mais direcionado, identificando e abordando dificuldades específicas dos alunos durante as atividades em grupo ou individuais (BERGAMI,2012).

A **Aprendizagem Baseada em Projetos (PBL)** é uma estratégia amplamente utilizada e eficaz para o ensino de **Pensamento Computacional** (KALELKAR et al., 2019 apud KIM, 2021). A **PBL** envolve os alunos na resolução de problemas complexos e autênticos do mundo real, permitindo-lhes construir conhecimento significativo por meio da investigação, cooperação e prática (MAO; CAO; HE, 2018 apud KIM, 2021). Essa abordagem fomenta a autonomia do aluno, a criatividade e a capacidade de resolver problemas (MAO; CAO; HE, 2018 apud KIM, 2021). Exemplos incluem a criação de modelos computacionais sobre furacões usando Scratch (SCRATCH FOUNDATION, [s.d.]) ou o design de aplicativos para necessidades sociais (PERKINS, 2012 apud KIM, 2021).

As atividades *unplugged* ensinam o **Pensamento Computacional** sem o uso de dispositivos digitais, utilizando materiais como papel, cartas ou atividades físicas (KIM; LEE, 2018 apud KIM, 2021). Estas atividades permitem que os alunos se concentrem nos conceitos subjacentes do **Pensamento Computacional** sem se prenderem à sintaxe de uma linguagem de programação (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021). Exemplos incluem "programar" um colega para realizar uma tarefa simples, usar mapas para ilustrar

a abstração (BARES et al., 2019 apud KIM, 2021) e a criação de fluxogramas da resolução do problema. O uso de atividades *unplugged* antes da programação baseada em computador pode levar a melhores resultados de aprendizagem (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021).

A **Aprendizagem Ativa e Colaborativa** é altamente incentivada (WANG; XING, 2016 apud KIM, 2021). Discussões em equipe, resolução de problemas em grupo e a comunicação de soluções alternativas são práticas que fomentam o **Pensamento Computacional** (EDTECHBOOKS, [s.d.]). A aprendizagem colaborativa, facilitada por tecnologias simples, pode melhorar as habilidades de resolução de problemas e o engajamento dos alunos (XU et al., 2020 apud KIM, 2021).

A **Abordagem Heurística e Baseada em Problemas** envolve apresentar problemas desafiadores e guiar os alunos a explorar e inovar, escalando gradualmente a complexidade, o que é indispensável para aprofundar a compreensão do **Pensamento Computacional** (KALELKAR et al., 2019 apud KIM, 2021). A abordagem socrática, na qual os conceitos são desenvolvidos por meio de questionamento dos alunos, também se mostra eficaz (WANG; XING, 2016 apud KIM, 2021).

No contexto dessas abordagens, o papel do professor se transforma de um mero transmissor de conhecimento para um facilitador,diagnosticador e curador do conhecimento. O educador deve atuar como gerador de questionamento, não fornecendo todas as soluções, mas guiando os alunos para que resolvam os problemas por si mesmos (MAO; CAO; HE, 2018 apud KIM, 2021). A capacidade do professor de diagnosticar dificuldades e oferecer orientação direcionada é determinante para o sucesso do processo de aprendizagem (MAO; CAO; HE, 2018 apud KIM, 2021). Isso parece indicar a necessidade de formação e desenvolvimento profissional contínuos para os educadores, capacitando-os não apenas no conhecimento do **Pensamento Computacional**, mas também nas estratégias pedagógicas para a sua implementação eficaz, incluindo a criação de ambientes de aprendizagem ativos e a oferta de retorno avaliativo construtivo.

## 5.2 Materiais Didáticos e Ferramentas

**Linguagens de Programação**: Python é frequentemente citada como uma linguagem de programação ideal para o ensino de **Pensamento Computacional**, especialmente em disciplinas introdutórias, devido à sua estrutura direta e facilidade de uso (UNIVERSITY OF PENNSYLVANIA, [s.d.]). Outras linguagens como OCaml, Java, C, C++ e Prolog também são utilizadas em currículos mais avançados (University of Cambridge, [s.d.]).

**Ambientes de Desenvolvimento e Ferramentas Visuais**: Ambientes como Scratch[1] (para crianças e iniciantes) (KIM; LEE, 2018 apud KIM, 2021), XLogo4Schools[2] e TigerJython, para Logo e Python, [^13] (MARCHAL; MOFFAT; VAN DER VEER, 2012 apud KIM, 2021) são empregados para reduzir a carga cognitiva e permitir que o foco recaia nos conceitos, notadamente para crianças e adolescentes. Além disso, ferramentas de depuração são essenciais para o aprendizado (Guzdial, 2015 apud KIM, 2021).

[^13]disponível em: TigerJython.

**Exemplos de Exercícios e Projetos encontrados na literatura**:

-**Decomposição**: quebrar a tarefa de fazer um sanduíche de geleia em etapas menores (EDTECHBOOKS, [s.d.]). - **Abstração**: criar um mapa de rotas de fuga da escola (EDTECHBOOKS, [s.d.]) ou usar mapas para simplificar o mundo (BARES et al., 2019 apud KIM, 2021). - **Algoritmização**: desenvolver o conjunto de instruções, passo a passo, necessário para atravessar uma rua com semáforo (ZHANG; WANG; ZHANG, 2019 apud KIM, 2021) ou para um robô fazer um sanduíche (EDTECHBOOKS, [s.d.]). - **Debugging**:

[1]disponível em: Scratch.
[2]disponível em: XLogo4Schools.

atividades que envolvem encontrar e corrigir erros em programas (EDTECHBOOKS, [s.d.]). ´**Projetos Integrados**: projetos em grupo que refletem a prática industrial (University of Cambridge, [s.d.]), design de estruturas usando software de engenharia (EDTECHBOOKS, [s.d.]), simulações de construção (EDTECHBOOKS, [s.d.]), e projetos que demonstram habilidades em ciência da computação como a criação e aplicação de algoritmos (University of Cambridge, [s.d.]).

**Recursos Didáticos Adicionais**: Livros como "Algorithms: A Problem Solving Journey" (DE KESEL; JANSKENS; DE LAET, 2019 apud KIM, 2021) e "Computer Science: A Problem-Solving Approach" (Guzdial, 2015 apud KIM, 2021) fornecem exemplos e explicações. Materiais com pseudocódigo e fluxogramas são utilizados para apoiar o aprendizado (KIM; LEE, 2018 apud KIM, 2021).

## 5.3 Avaliação da Aplicação da Metodologia DAAD no Desempenho dos Alunos

A avaliação das habilidades de **Pensamento Computacional** é um aspecto indispensável para medir o desempenho dos alunos e a eficácia das metodologias de ensino. No entanto, este é um campo em evolução, apresentando desafios relacionados à falta de consenso na definição do **Pensamento Computacional** e à necessidade de instrumentos validados. A natureza complexa e multifacetada do **Pensamento Computacional**, que abrange aspectos cognitivos, atitudinais e práticos, exige que a avaliação vá além dos testes de conhecimento tradicionais (KIM; LEE, 2018 apud KIM, 2021).

Uma combinação de exames, avaliação de projetos, rubricas e autoavaliações é essencial para capturar a gama completa de habilidades de **Pensamento Computacional** (*unplugged*, [s.d.]). Indicando que as instituições de ensino superior devem desenvolver e implementar abordagens de avaliação abrangentes que reflitam a profundidade e a amplitude do **Pensamento Computacional**, investindo em design de rubricas detalhadas e incentivando projetos práticos que exijam a aplicação de múltiplos componentes de **Pensamento Computacional**.

### 5.3.1 Métodos de Avaliação de Habilidades de Pensamento Computacional/DAAD

**Testes e Exames**: testes de múltipla escolha e questões de resposta curta são utilizados para medir o conhecimento e a compreensão de conceitos de **Pensamento Computacional** (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021). Alguns testes, como o "Computational Thinking Performance Test"[3], focam em pensamento lógico, generalização e abstração (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021). A Figure 5.1 exemplifica o tipo de testes de avaliação de **Pensamento Computacional** que podem ser aplicados, com questões que avaliam a capacidade de decomposição, abstração e algoritmização.

[3]em tradução livre: Teste de Desempenho de Pensamento Computacional.

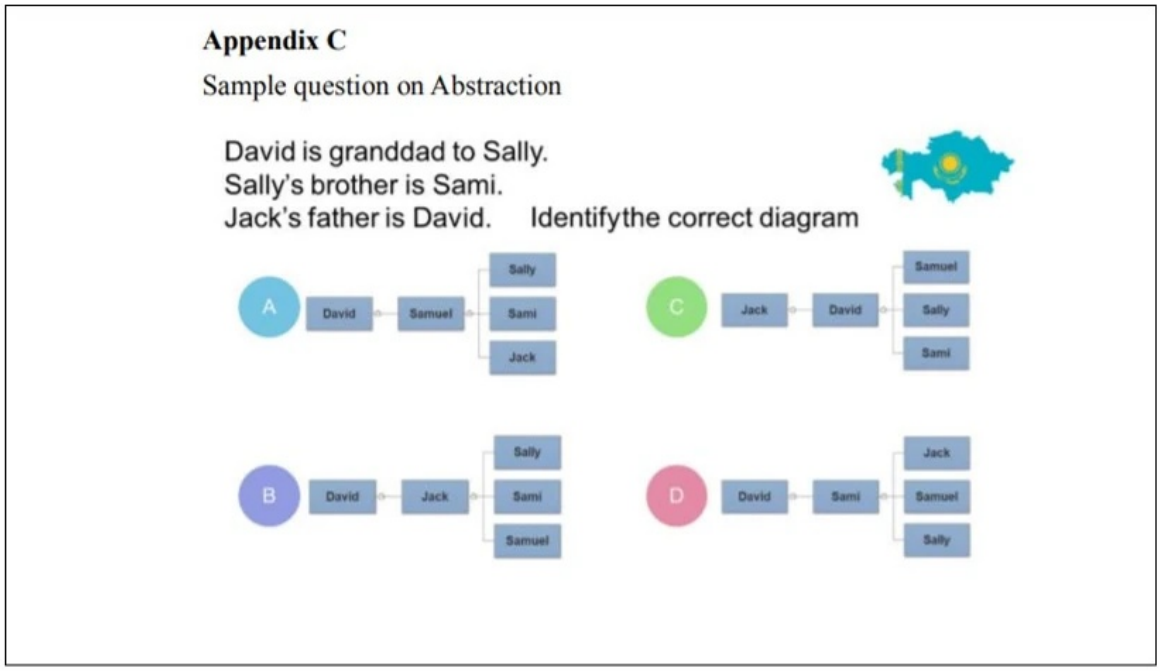


Figure 5.1: Exemplo de Testes de Avaliação de **Pensamento Computacional**(GROVER & COENRAAD, 2021)

**Avaliação de Projetos e Portfólios**: a avaliação de projetos é util para discernir como os alunos aplicam seus conhecimentos para pensar computacionalmente e projetar artefatos computacionais criativos (YANG et al., 2018 apud KIM, 2021). Portfólios de projetos e tarefas relacionadas são utilizados para documentar o aprendizado (*unplugged*, [s.d.]). Ao contrário dos testes e exames, a avaliação de portifólio requer avaliação manual ou com o uso de ferramentas de inteligência artificial.

**Rubricas**: rubricas são ferramentas valiosas para guiar os professores na avaliação de atividades e objetivos de **Pensamento Computacional**, articulando critérios com descritores de desempenho que demonstram níveis progressivamente mais sofisticados de domínio (KIM; LEE, 2018 apud KIM, 2021). A Universidade de Delaware, por exemplo, desenvolveu uma rubrica para **Pensamento Computacional** com quatro dimensões: decomposição, algoritmos, dados e abstração (KIM; LEE, 2018 apud KIM, 2021). A Table 5.1 apresenta um exemplo de rubrica para avaliação de habilidades de **Pensamento Computacional**.

Table 5.1: Exmeplo de Rubrica para Avaliação de Habilidades de **Pensamento Computacional** (COMPUTER CIENCE TEACHER ASSOCIATION, 2011).

| Critérios | Capstone (4) | Marco de Referência (3) (2) (1) |
|---|---|---|
| **Decomposição do Problema** Dividir tarefas em partes menores e gerenciáveis | Identifica um problema de grande escala. Divide o problema em partes menores e gerenciáveis. Identifica variáveis do problema e omite variáveis estranhas. Determina quais variáveis são controláveis e quais são determinadas por fatores externos. | |
| **Abstração** Reduzir a complexidade para definir a ideia principal | Identifica características de algo para reduzir a um conjunto de características essenciais. Encontra semelhanças e desconsidera diferenças sem importância em processos ou objetos. Reduz um conjunto de dados a uma representação simplificada (modelo). | |

| Critérios | Capstone (4) | Mandatório (3) | Marco de Referência (2) (1) |
|---|---|---|---|
| **Algoritmos** Série de passos ordenados tomados para resolver um problema ou atingir um objetivo | Desenvolve uma sequência de passos ou instruções para realizar uma tarefa ou resolver um problema. Refina um algoritmo com base em testes sob diferentes cenários (entradas) para correção e eficiência. | | |
| **Automação** Habilitar computadores ou máquinas para realizar tarefas repetitivas ou tediosas | Identifica oportunidades e benefícios da automação; Usa ferramentas automatizadas para realizar tarefas rotineiras; Articula os prós e contras da automação na sociedade; Cria um artefato que envolve automação (ex: um programa, máquina, modelo de máquina). | | |

| Critérios | Capstone (4) | Marco (3) | Marco de Referência (2) | (1) |
|---|---|---|---|---|
| **Simulação** Representação ou modelo de um processo. Também envolve a execução de experimentos usando modelos | Cria uma simulação de um processo usando software, animação, pessoas, objetos, planilha ou outro meio apropriado. | | | |

| Critérios | Capstone (4) | Marco de Referência (3) (2) (1) |
|---|---|---|
| **Paralelização** Organizar recursos para realizar tarefas simultaneamente para atingir um objetivo comum | **Elização** Dado um projeto com muitas subtarefas, identifica tarefas que precisam ser feitas sequencialmente e tarefas que podem ser feitas simultaneamente, e quando os pontos de verificação precisam ser feitos e as coisas reunidas para cumprir os prazos de forma eficiente. | Mantém |
| **Coleta de Dados** Reunir informações apropriadas | Dado um problema, projeto ou estudo, determina as informações apropriadas a serem coletadas e é capaz de identificar informações estranhas e inúteis para coletar. Desenha instrumentos (ex: pesquisas, experimentos) para coletar os dados apropriados (qualitativos ou quantitativos). | |

| Critérios | Capstone (4) | Marco (3) | Referência (2) | Marco (1) |
|---|---|---|---|---|
| **Representação de Dados** Determinar representações eficazes | Destaca representações eficazes (ex: tabelas, gráficos, diagramas) ou visualizações para ajudar a extrair conclusões significativas de um conjunto de dados brutos. | | | |

| Critérios | Capstone (4) | Marco de Referência (3) (2) (1) |
|---|---|---|
| **Análise de Dados** Usando visualização de dados ou ferramentas estatísticas, identifica e descreve tendências e padrões em um conjunto de dados, e testa hipóteses ou conclusões propostas. | | |

**Autoavaliação e Pesquisas**: escalas psicométricas e questionários de autoavaliação são empregados para medir a percepção dos alunos sobre suas próprias habilidades de **Pensamento Computacional** e sua autoeficácia (BAO et al., 2019 apud KIM, 2021).

**Observação e Entrevistas**: A observação em sala de aula e as entrevistas, individuais ou em grupo, são técnicas de avaliação formativa que fornecem percepções sobre o engajamento dos alunos no **Pensamento Computacional** e suas estratégias de resolução de problemas (BAO et al., 2019 apud KIM, 2021).

## 5.4 Ferramentas e Instrumentos de Avaliação

**UniCTCheck (CTScore e CTProg)**: um método inovador para avaliar habilidades de **Pensamento Computacional** em alunos universitários de Ciência da Computação. O CTScore é um aplicativo web interativo que mede sete componentes de **Pensamento Computacional**, reconhecimento de padrões, pensamento criativo, pensamento algorítmico, resolução de problemas, pensamento crítico, decomposição e abstração, por meio de 12 questões. O CTProg é uma escala psicométrica que mede a compreensão conceitual de cinco fundamentos de programação, direções básicas e sequências, condicionais, laços de repetição, funções, estruturas de dados (KIM; LEE, 2018 apud KIM, 2021).

**Bebras Questions**: utilizadas como ferramenta de teste para analisar melhorias nas capacidades de **Pensamento Computacional** para crianças e adolescentes (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021).

**Dr. Scratch, LOGO, C, C++, AgentCubes**: linguagens, ferramentas de depuração, e teste mencionadas como ferramentas de avaliação que podem ser utilizadas para aprimorar o **Pensamento Computacional**, com foco principal no treinamento de habilidades de programação (MAO; CAO; HE, 2018 apud KIM, 2021).

Apesar do crescimento na pesquisa sobre avaliação de **Pensamento Computacional**, ainda existe uma falta de consenso na descrição ou nas partes constituintes do **Pensamento Computacional**, o que dificulta a padronização e validação de instrumentos de avaliação (SUNG et al., 2017 apud KIM, 2021). Muitos estudos apontam para a necessidade de mais pesquisas para desenvolver e validar uma conceituação teórica robusta do **Pensamento Computacional** e, consequentemente, ferramentas de avaliação mais eficazes (SUNG et al., 2017 apud KIM, 2021). Para o avanço da educação em **Pensamento Computacional**, é importante que a comunidade acadêmica continue a colaborar na definição e operacionalização do **Pensamento Computacional**, o que, por sua vez, permitirá o desenvolvimento de instrumentos de avaliação mais rigorosos e comparáveis. Isso é indispensável para medir o impa**Pensamento Computacional**o das intervenções pedagógicas e garantir a qualidade da educação em **Pensamento Computacional**.

## 5.5 Melhores Práticas e Desafios na Implementação de Metodologias de Resolução de Problemas

A implementação de metodologias de resolução de problemas, como o **Pensamento Computacional** e seus componentes **DAAD**, no ensino superior apresenta tanto oportunidades significativas quanto obstáculos consideráveis. Existe uma lacuna notável entre a teoria e a prática na implementação do **Pensamento Computacional**. Embora a literatura acadêmica amplamente reconheça a importância e os benefícios do **Pensamento Computacional** (EDTECHBOOKS, [s.d.]), os desafios comuns revelam uma dificuldade significativa em traduzir essa importância em práticas educacionais eficazes e generalizadas (KIM; LEE, 2018 apud KIM, 2021). **A falta de clareza na definição do** Pensamento Computacional**, a preparação inadequada dos professores e a dificuldade de integração curricular são barreiras recorrentes que impedem a adoção em larga escala**.

### 5.5.1 Melhores Práticas Identificadas

O **Pensamento Computacional** é mais eficaz quando integrado em diversas disciplinas, não apenas na ciência da computação (WANG; XING, 2016 apud KIM, 2021). Essa abordagem permite que os alunos apliquem o **Pensamento Computacional** a problemas relevantes para suas áreas de estudo, desde a análise de dados históricos até a otimização de processos logísticos (WANG; XING, 2016 apud KIM, 2021). **A contextualização dos problemas em cenários do mundo real ou em suas próprias pesquisas aumenta o engajamento dos alunos** (KALELKAR et al., 2019 apud KIM, 2021).

O ensino integral, voltado para a formação do indivíduo presupõe que. além das habilidades técnicas, o ensino de **Pensamento Computacional** deve cultivar atitudes como confiança na resolução de problemas, persistência diante de desafios, tolerância a diferenças e capacidade de colaborar (EDTECHBOOKS, [s.d.]). A aprendizagem baseada em projetos e atividades *unplugged* são eficazes para fomentar essa mentalidade (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021).

Do ponto de vista do profissional da educação, será indispensável garantir que os professores tenham um alto nível de conhecimento e prontidão em **Pensamento Computacional** (MAO; CAO; HE, 2018 apud KIM, 2021). Programas de desenvolvimento profissional que oferecem ferramentas e ideias aprofundadas para a integração do **Pensamento Computacional** são essenciais (PARK; KIM, 2018 apud KIM, 2021). Isso inclui o apoio aos professores para criar novos materiais instrucionais e adaptar suas estratégias pedagógicas (MAO; CAO; HE, 2018 apud KIM, 2021).

Finalmente, introduzir conceitos de **Pensamento Computacional** de forma gradual, revisitando e expandindo a compreensão ao longo do currículo, é uma prática eficaz (MARCHAL; MOFFAT; VAN DER VEER, 2012 apud KIM, 2021). Começar com atividades *unplugged* e progredir para desafios de programação *plugged* pode otimizar o aprendizado (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021).

### 5.5.2 Desafios Comuns

O primeiro desafio surge da falta de consenso na definição e operacionalização do ensino de **Pensamento Computacional**. Neste cenário, uma das maiores dificuldades é a ausência de uma definição única e universalmente aceita de **Pensamento Computacional** (*unplugged*, [s.d.]). Essa ambiguidade afeta o design do currículo, a seleção de tópicos e, principalmente, a avaliação das habilidades dos alunos. Além disso, muitos educadores carecem de compreensão, confiança e habilidades necessárias para implementar o **Pensamento Computacional** de forma eficaz (KIM; LEE, 2018 apud KIM, 2021). A expectativa de que as habilidades de **Pensamento Computacional** se desenvolvam naturalmente em alunos de ciência da computação é muitas vezes inadequada e irreal (ZHANG et al., 2018 apud KIM, 2021).

A integração do **Pensamento Computacional** em currículos já estabelecidos, especialmente em disciplinas não diretamente relacionadas com ciência, matemática e as engenharias, pode ser um desafio devido à falta de uma abordagem padronizada (TSAI; TSAI, 2018 apud KIM, 2021). Alguns conceitos e práticas de **Pensamento Computacional**, como condicionais, dados, operadores, testes e depuração, e abstração/modularização, são identificados como difíceis de aprender para alunos iniciantes (LIU et al., 2019 apud KIM, 2021). Isso pode levar a uma falta de engajamento e persistência dos alunos (LIU et al., 2019 apud KIM, 2021).

A avaliação das capacidades de realização do **Pensamento Computacional** é complexa e as ferramentas existentes são limitadas, muitas vezes focando em percepção-atitude ou testes de múltipla escolha simples (KIM; LEE, 2018 apud KIM, 2021). Há uma necessidade de pesquisa e implantação de instrumentos mais rigorosos e validados (SUNG et al., 2017 apud KIM, 2021).

Apesar dos desafios, a própria natureza do **Pensamento Computacional**, que enfatiza a resolução de problemas, a criatividade e a colaboração (EDTECHBOOKS, [s.d.]), impulsiona a adoção de pedagogias inovadoras como a aprendizagem baseada em projetos e atividades *unplugged* (RODRIGO; CABRERA; BARRERA, 2018 apud KIM, 2021). Além disso, a aplicabilidade universal do **Pensamento Computacional**, incentiva a colaboração entre diferentes departamentos e disciplinas (WANG; XING, 2016 apud KIM, 2021). Esta universalidade se dá porque o **Pensamento Computacional** não é apenas um conjunto de habilidades a ser ensinado, mas uma filosofia que pode transformar a própria abordagem pedagógica nas universidades. Ao invés de ser visto como um fardo adicional, o ensino do **Pensamento Computacional** pode ser uma oportunidade para as instituições de ensino superior inovarem em suas metodologias, promoverem a aprendizagem ativa e prepararem os alunos de forma mais holística para os desafios complexos do século XXI, fomentando uma cultura de resolução de problemas e inovação em todo o campus.

# 6 Projeto de Disciplina de Raciocínio Algorítmico

Uma disciplina de **Raciocínio Algorítmico**, nomeada assim para que exista aderência os conceitos de **Pensamento Computacional** e da metodologia **DAAD** e, principalmente, por ser a terminologia utilizada na Pontifícia Universidade Católica do Paraná. Este será o projeto uma disciplina de 80 horas-aula que podem ser distribuídas de formas diferente, dependendo do formato acadêmico. Pode ser uma disciplina intensiva de aproximadamente 5 semanas, com cerca de 16 horas/semana, ou uma disciplina padrão de um semestre, com 4-6 horas/semana ao longo de 14-18 semanas. Usando um modelo de 16 semanas com 4h aula por semana e 16h de trabalhos acadêmicos a serem executados fora de sala o que equivale a aproximadamente $1,72$ nos EUA, alguma coisa entre $2,7$ a $3,2$ **ECTS* e, em média $3,56$ na China.

A estrutura do currículo foi projetada para construir progressivamente as habilidades **DAAD**, começando com problemas e exercícios básicos e, paulatinamente, avançando para aplicações mais complexas. A limitação de 80 horas, embora substancial, exige um equilíbrio cuidadoso entre a amplitude e a profundidade do conteúdo. Os iniciativas das universidades estudadas demonstram que o **Pensamento Computacional** pode ser uma disciplina autônoma, como em Cardiff, ETH Zurich, Peking, ou integrado em várias disciplinas centrais da Ciência da Computação, como no Imperial College, Oxford, Penn State. **O desafio reside em cobrir os elementos** DAAD** com profundidade suficiente para graduandos, ao mesmo tempo em que se introduzem aplicações práticas**.

A progressão de conceitos para aplicações, e de problemas mais simples para mais complexos, parece ser uma estratégia eficaz. Isso implica que, embora conceitos amplos sejam introduzidos, a profundidade é alcançada por meio da aplicação repetida e da crescente complexidade dos projetos. Portanto, **o currículo deve priorizar a prática intensiva sobre a cobertura teórica exaustiva, garantindo que os alunos pratiquem o Pensamento Computacional em vez de apenas aprenderem sobre ele**. Dessa forma, em uma situação ideal, as 80 horas devem ser predominantemente dedicadas a sessões de laboratório, resolução de problemas e trabalho em projetos.

No mercado brasileiro, usando como referência apenas a experiência do autor na Pontifícia Universidade Católica do Paraná e em outras universidades da Cidade de Curitiba, os alunos dos turnos noturnos trabalham durante o dia e não têm a mesma disponibilidade de tempo dos alunos das universidades do EUA que foram analisadas nesta pesquisa. Portanto, caberá ao professor da disciplina analisar sua situação particular e optar pelo uso, ou não, da metodologia **Sala de Aula Invertida**. O não uso desta metodologia terá impacto na criação dos constructos necessários a criação das competências relacionadas ao **DAAD** já que o tempo dedicado a resolução de problemas, exercícios e projetos em sala será reduzido, graças a necessidade da apresentação da teoria. Mantendo essa limitação de tempo fora de sala, os exercícios apresentados nos 10 apêndices deste texto foram concebidos para limitar a necessidade de apresentação de teoria.

Finalmente, o autor optou pela **Linguagem C++ 23** em vez do Python, utilizado pela maioria das disciplinas estudadas. Esta escolha é proposital para que o aluno possa criar em código, exatamente a mesma estrutura de solução que ele terá criado em fluxograma. Isso é necessário para garantir o progresso na criação de complexidade. O aluno irá criar fluxogramas simples, usando apenas os módulos Início, Atribuição, Decisão e Fim. Esta

estrutura exige que a linguagem de programação escolhida possa utilizar `goto`. O Python não possui este comando nem os artefatos de compilação necessários para sua implementação.

A seguir, a proposta de conteúdo, agenda e alocação de tempo para a disciplina **Raciocínio Algorítmico**. A proposta é dividida em quatro módulos principais, cada um com foco em um componente do **Pensamento Computacional**: Decomposição, Abstração, Algoritmização e Depuração. Cada módulo tem a sua própria duração e está projetado para ser interativo, com atividades práticas e projetos que reforçam os conceitos aprendidos. Completando as atividades em sala já estão definidos 3 atividades de pesquisa e solução de problemas para serem realizados fora de sala que poderão, a critério do professor, ser usados como forma de avaliação.

## 6.1 Módulo 1: Semanas 1-3 (12 Horas-Aula): Introdução ao Pensamento Computacional e Decomposição

Este módulo foca na compreensão conceitual do **Pensamento Computacional** e sua importância, explorando a Decomposição segundo a definição "como quebrar problemas complexos em subproblemas menores e identificáveis" (FRONTIERS, 2022).

1. **Atividades Desplugadas (`unplugged`)**: serão utilizadas atividades sem computador para ilustrar a decomposição. Exemplos incluem "fazer um bolo", "fazer um sanduíche de presunto e queijo" ou "Programar o Professor". A descrição destas atividades está disponível no Chapter 8.

2. **Fluxogramas e Algoritmos Simples**: introdução às técnicas de fluxogramas para a definição de algoritmos e solução de problemas simples. Serão realizados exercícios de algoritmos na forma de fluxogramas, seguindo o padrão IEEE e apenas os módulos referentes a Inicio, Fim, Atribuição e Decisão aplicando diretamente os princípios de decomposição. Estes exercícios estão disponíveis no **?@sec-apendice2**.

3. **Estruturas Sintáticas Modulares**: introdução aos conceitos de artefatos sintáticos para laços de repetição (`for`, `while`, `do while`) e funções, na forma de módulos reutilizáveis criados por blocos de fluxogramas. Exercícios de fluxogramas para induzir a reutilização de conjuntos de blocos. Estes exercícios estão disponíveis no **?@sec-apendice3**. A Figure 6.2 resume o padrão IEEE e as estruturas básicas e os blocos necessários para `for`, `while`, `do while`.

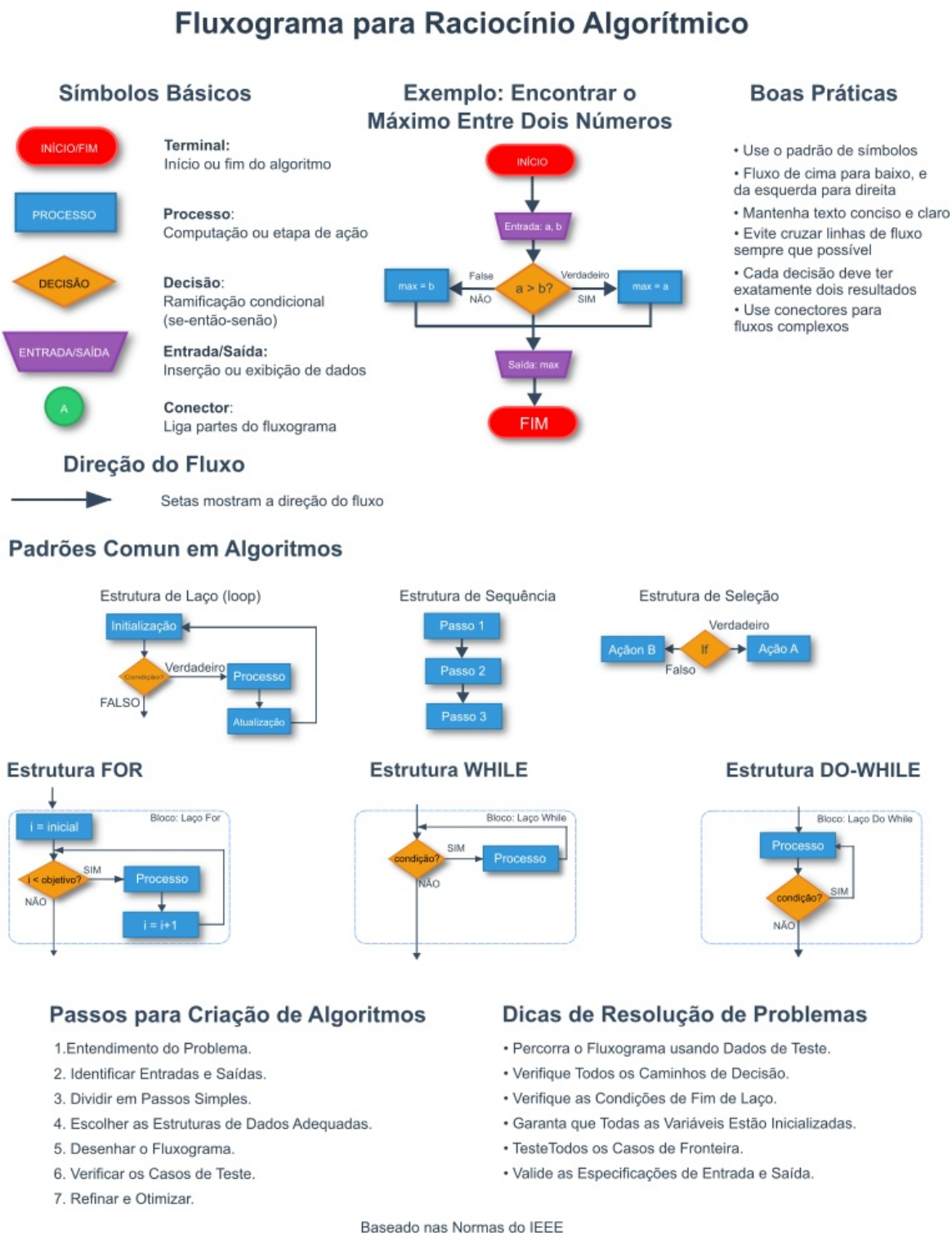


Figure 6.1Resumo do Padrão IEEE para Fluxogramas

Figure 6.2

## 6.2 Módulo 2: Semanas 4-8 (20 Horas-Aula): Abstração e Reconhecimento de Padrões

Nesta fase, embora o reconhecimento de padrões não esteja explicitamente no acrônimo **DAAD**, esta técnica é um elemento essencial do **Pensamento Computacional** e da resolução de problemas, frequentemente agrupado com abstração em ferramentas de **Pensamento Computacional** (FRONTIERS, 2022). Isso implica a necessidade de ensinar a compreensão da abstração: identificar informações essenciais, ignorar detalhes irrelevantes e criar modelos simplificados (FRONTIERS, 2022). Exercícios de Fluxograma mais complexos e livres de padrões serão propostos para induzir a criatividade, além de fomentar a capacidade de reconhecimento de padrões, ou seja, a capacidade de identificar similaridades, tendências e generalizações (FRONTIERS, 2022).

1. **Introdução à Linguagem C++ (Compatibilidade C)**: Introdução à Linguagem de Programação C++, obrigatoriamente em forma de compatibilidade com a Linguagem C, usando indispensavelmente a palavra-chave `goto` para refazer exatamente os mesmos exercícios aplicados na Section 6.1.

2. **Abstração na Programação com C++**: Aplicação da abstração na programação: criação de componentes reutilizáveis, funções, laços de repetição e estruturas de dados (PENN STATE, [s.d.]). Utilizando a Linguagem C++ com as abstrações de função, `for`, `while`, `do while`, para refazer exatamente os mesmos exercícios aplicados na Section 6.1.

3. **Estruturas de Dados Simples**: Definição e aplicação de estruturas de dados simples como `vectors`, `arrays` e `structs`. Neste ponto, serão aplicados os primeiros exercícios complexos. Os alunos ainda precisarão fazer o fluxograma e validar a solução antes de desenvolver o código. Estes exercícios estão disponíveis no **?@sec-apendice4**.

4. **Exercícios Avançados de Interpretação**: Serão propostos exercícios avançados de visualização de dados (DIGITAL PROMISE, 2022), simulação de sistemas com modelos simplificados (LEHMANN, 2024) e identificação de padrões de código recorrentes para refatoração. Problemas que requeiram interpretação para encontrar as soluções de decomposição e abstração.

5. **Trabalho Extraclasse 1**: os alunos deverão desenvolver o projeto de um sistemas especialista simples. A entrega, com correção automática e análise por inteligência artificial, deverá ser feita em um repositório público, como o GitHub na aula da semana 10. Os enunciados destes sistemas especialistas estão disponíveis no **?@sec-apendice5**.

## 6.3 Módulo 3: Semanas 9-13 (20 Horas-Aula): Algoritmização e Estruturas de Dados

Nesta fase, o foco será no Design de Algoritmos, priorizando o desenvolvimento de procedimentos passo a passo, análise de eficiência e complexidade algorítmica (FRONTIERS, 2022).

1. **Algoritmos Comuns**: Estudo de algoritmos comuns: busca (linear, binária), ordenação (`bubble sort`, `quick sort`) e algoritmos básicos de grafos (`transversalidade`) (PENN STATE, [s.d.]). Sem a matemática envolvida. Porém, forçando a análise dos custos computacionais, complexidade, relacionada a estes algoritmos. O professor deve introduzir o conceito de pseudocódigo, usando como comparação os fluxogramas já vistos. Os pseudocódigos e fluxogramas das estruturas propostas estão disponíveis no **?@sec-apendice6**.

2. **Estruturas de Dados**: listas, árvores, grafos e tabelas *hash* (PENN STATE, [s.d.]). A partir deste ponto, os alunos devem ser escolher entre fluxogramas, ou pseudocódigo, para definir as estruturas de dados antes de implementar o código correspondente em C++. Todos os entregáveis devem incluir o pseudocódigo ou o fluxograma, além da implementação em C++ em um ambiente online. Os exercícios de estruturas de dados estão disponíveis no **?@sec-apendice7**.

3. **Tarefas de Programação e Análise**: Tarefas de programação que exigem a implementação e análise de algoritmos. Serão propostos exercícios que requeiram interpretação para encontrar as soluções de decomposição, abstração e algoritmização. A partir deste ponto, o professor deve introduzir o uso de pseudocódigo e deixar o uso, ou não, de fluxogramas a critério dos alunos. Estes exercícios estão disponíveis no **?@sec-apendice8**.

4. **Trabalho Extraclasse 2**: os alunos deverão desenvolver o projeto de um sistema especialista mais complexo, que exija a aplicação de algoritmos e estruturas de dados. A entrega, com correção automática e análise por inteligência artificial, deverá ser feita em um repositório público, como o GitHub na aula da semana 16. Os enunciados destes sistemas especialistas estão disponíveis no **?@sec-apendice9**.

## 6.4 Módulo 4: Semanas 14-16 (12 Horas-Aula): Depuração e Aplicação em Projetos

Esta etapa final se concentrará na Depuração Sistemática: identificação de erros, técnicas de teste e rastreamento de execução de programas (FRONTIERS, 2022). Este é o fim do `std::cout`.

1. **Princípios de Teste de Software**: Serão abordados os princípios de teste de software (teste de unidade, teste de integração) (FRONTIERS, 2022). De forma teórica, mas com exercícios práticos de depuração.

2. **Sistemas de Controle de Versão**: Introdução a sistemas de controle de versão (ex: Git no GitHub) para desenvolvimento colaborativo e rastreamento de erros.

3. **Trabalho Extraclasse 3**: Os alunos desenvolverão um Projeto, que exigirá a aplicação integrada de todos os elementos **DAAD** para resolver um problema maior e mais complexo do mundo real, com forte ênfase no desenvolvimento iterativo e na depuração contínua (LEHMANN, 2023). O enunciado deste projeto está disponível no **?@sec-apendice10**. A entrega, com correção automática e análise por inteligência artificial, deverá ser feita em um repositório público, como o GitHub na aula da semana 18.

## 6.5 Objetivos de Aprendizagem Essenciais

Ao final da disciplina, os alunos deverão ser capazes de:

- **Decomposição**: decompor problemas computacionais complexos em subproblemas menores e gerenciáveis, identificando interfaces claras entre eles.
- **Abstração**: identificar características essenciais e informações relevantes de um problema, criando modelos ou representações generalizadas, e aplicar múltiplos níveis de abstração para gerenciar a complexidade.
- **Algoritmização**: projetar procedimentos passo a passo eficientes e corretos, algoritmos, para resolver problemas decompostos, e expressá-los usando pseudocódigo e uma linguagem de programação escolhida.

- **Depuração**: identificar, localizar e corrigir sistematicamente erros em seus próprios códigos e nos de outros, empregando diversas estratégias de depuração e teste.
- **Resolução de Problemas**: aplicar a metodologia **DAAD** de forma iterativa para resolver desafios computacionais e de engenharia novos, demonstrando pensamento analítico.
- **Colaboração**: Trabalhar eficazmente em equipes em projetos de programação, utilizando ferramentas e práticas colaborativas.
- **Comunicação**: Articular claramente definições de problemas, soluções algorítmicas e processos de depuração para públicos técnicos e não técnicos.

# 7 Considerações Finais

O **Pensamento Computacional**, criado a partir da aplicação sistemática de seus componentes, **D**ecomposição, **A**bstração, **A**lgoritmização e **D**ebugging, **DAAD**, é uma habilidade essencial e cada vez mais reconhecida na educação superior em cursos de Ciência e Engenharia da Computação, e além. A pesquisa parece demonstrar uma clara transição curricular, na qual o foco se expande da mera proficiência em programação para **uma compreensão mais profunda e ampla do raciocínio computacional**. A interconexão dos componentes **DAAD** forma um ciclo iterativo de resolução de problemas, da natureza transdisciplinar do **Pensamento Computacional** permitindo sua aplicação em diversas áreas do conhecimento, posicionando-o como uma competência universal para o século XXI.

As universidades globais, incluindo instituições de prestígio nos EUA, Reino Unido, Europa e China, estão integrando o **Pensamento Computacional** em seus currículos, muitas vezes com disciplinas dedicadas e projetos práticos. Estratégias pedagógicas como a Aprendizagem Baseada em Projetos e atividades *unplugged* são amplamente utilizadas para promover o engajamento ativo e o desenvolvimento de uma mentalidade computacional. A avaliação das habilidades de **Pensamento Computacional**, embora complexa, está evoluindo para abordagens multimodais que combinam testes, rubricas e avaliação de projetos. No entanto, a implementação do **Pensamento Computacional** no ensino superior enfrenta desafios significativos, como a falta de uma definição padronizada, a preparação inadequada de professores e as dificuldades na integração curricular.

Além das diferenças de carga horária em sala existem diferenças na forma como as universidades avaliam a importância do **Pensamento Computacional** e do raciocínio algorítmico. O MIT, por exemplo, considera um requisito mínimo de computação para todos os seus estudantes de graduação (MIT, [s.d.]) independente do curso. Ressaltando a importância atribuída ao **Pensamento Computacional**. A adoção deste grau de importância implica em uma redefinição do que constitui uma educação superior completa na era digital, visando preparar os estudantes para um futuro no qual a capacidade de analisar, decompor e resolver problemas de forma sistemática, mesmo sem um computador, será tão valiosa quanto a leitura e a escrita. Esta abordagem também abre caminho para uma maior colaboração interdisciplinar no desenvolvimento de currículos.

Para um currículo de 80 horas, isso implica a incorporação de problemas autênticos que ressoem com os interesses dos alunos ou com a relevância social, em vez de desafios de codificação puramente abstratos. A aprendizagem baseada em projetos, como exemplificado no currículo de robótica da CMU, é uma adequação natural.

A presença de módulos dedicados ao **Pensamento Computacional** em universidades como Cardiff (CARDIFF UNIVERSITY, [s.d.]), e a estrutura de especializações como a da Universidade de Glasgow, revelam uma importante estratégia: o **Pensamento Computacional** pode ser introduzido em vários níveis de complexidade e integrado de forma progressiva. As atividades *unplugged* (NEW YORK HALL OF SCIENCE, [s.d.]) mencionadas em outras fontes reforçam essa ideia, fornecendo um ponto de entrada de baixa complexidade antes de avançar para atividades *plugged* mais complexas.

A projeção destes conceitos em um currículo de 80 horas, se traduz em uma abordagem por fases, na qual os conteúdos iniciais devem empregar ferramentas mais simples ou métodos *unplugged* para construir a compreensão conceitual, permitindo uma transição gradual para ambientes de programação mais complexos e projetos maiores à medida que os alunos progridem.

As estratégias da ETH Zurich se forem adaptadas para o currículo de 80 horas, implicarão em um modelo de aprendizagem híbrido, que aproveita recursos online para os conceitos fundamentais e dedica o tempo presencial a sessões colaborativas de resolução de problemas, trabalho em projetos e depuração guiada, pode ser altamente benéfico.

china: Essa amplitude de escopo é uma tendência observada, sugerindo que o PC é visto como inseparável da codificação prática e do design de sistemas. Para um currículo de 80 horas, isso significa buscar essa integração abrangente, garantindo que os princípios **DAAD** sejam ensinados *através* da programação e da resolução de problemas, em vez de isoladamente, reforçando a relevância prática do PC.

Apesar dos obstáculos discutidos neste trabalho, o **Pensamento Computacional** atua como um catalisador para a inovação pedagógica e a colaboração interdisciplinar, impulsionando as instituições a adotarem metodologias de ensino mais ativas e a prepararem os alunos de forma mais abrangente para os desafios complexos do mundo moderno.

## 7.1 Recomendações para o Design Curricular, Estratégias Pedagógicas e Avaliação no Ensino Superior de Ciência e Engenharia da Computação

1. **Design Curricular**: o projeto do currículo de formação do aluno deve ser centrado no aluno, com foco na construção de habilidades de **Pensamento Computacional** por meio da prática e da aplicação em problemas do mundo real. A integração dos componentes **DAAD** deve ser feita de forma progressiva, começando com conceitos básicos e avançando para aplicações mais complexas.

    - **Formalização da Integração**: as universidades devem formalizar a integração dos componentes **DAAD** em disciplinas introdutórias e avançadas, tanto para estudantes de Ciência e Engenharia da Computação quanto para todas as áreas diretamente relacionadas, nos estudos de ciência, matemática e engenharia e mesmo em áreas tipicamente distantes como direto e medicina. Isso garante que todos os alunos desenvolvam uma base sólida de **Pensamento Computacional**. E possam usufruir das estruturas cognitiva para a solução de problemas reais.
    - **Currículos Modulares e Flexíveis**: desenvolver currículos modulares que permitam flexibilidade e adaptação a diferentes cargas horárias. Um módulo dedicado de aproximadamente 80 horas-aula, focado nos fundamentos e aplicações práticas do **Pensamento Computacional**, pode ser um formato eficaz para aprofundar essas habilidades em cursos de Ciência e Engenharia da Computação. Mas pode ser excessivo para outras áreas do conhecimento.
    - **Incentivo à Interdisciplinaridade**: promover ativamente a integração interdisciplinar do **Pensamento Computacional**, criando oportunidades para que os alunos apliquem o raciocínio computacional em problemas relevantes para diversas áreas do conhecimento, demonstrando a universalidade e o valor prático dessas habilidades.

2. **Estratégias Pedagógicas**: o conjunto de estratégias pedagógicas adequadas ao ensino de **Pensamento Computacional** é, como vimos, diverso. Contudo, podemos destacar a necessidade de:

    - **Priorização da Aprendizagem Ativa**: priorizar metodologias de aprendizagem ativa, como Aprendizagem Baseada em Projetos e Aprendizagem Baseada em Problemas, para promover a aplicação prática do **Pensamento Computacional** e o desenvolvimento de uma mentalidade de resolução de problemas e perseverança.

- **Abordagem Progressiva**: utilizar uma abordagem didática progressiva, começando com atividades *unplugged* para construir a compreensão conceitual dos componentes **DAAD** antes de introduzir as atividades *plugged*. Essa transição gradual deve ajudar a solidificar o aprendizado e a reduzir barreiras iniciais.
- **Capacitação Docente Contínua**: investir em programas de desenvolvimento profissional contínuo para os professores, focando tanto no aprofundamento do conhecimento em **Pensamento Computacional** quanto nas estratégias pedagógicas eficazes para seu ensino. Os professores devem ser capacitados para atuar como facilitadores e diagnosticadores, guiando os alunos em vez de apenas transmitir informações.

**Avaliação**: apesar de, neste trabalho, não termos abordado a avaliação de forma mais aprofundada, é importante ressaltar que a avaliação do **Pensamento Computacional** deve ser abrangente e multidimensional, refletindo a complexidade das habilidades envolvidas. Adotar abordagens de avaliação multimodal que combinem testes de conhecimento, avaliação de projetos com rubricas detalhadas e autoavaliações. Essa combinação permite capturar a complexidade das habilidades de **Pensamento Computacional** em suas diversas dimensões, cognitiva, prática e atitudinal. Finalmente, as instituições devem contribuir para a pesquisa em avaliação de **Pensamento Computacional**, buscando o desenvolvimento de instrumentos validados e a padronização de métricas. Isso é indispensável para permitir comparações entre diferentes abordagens pedagógicas e para o aprimoramento contínuo da educação em Pensamento Computacional.

## 7.2 Direções Futuras para o DAAD

Para consolidar o modelo:

Testes empíricos em cursos de engenharia de software (ex.: análise comparativa DAAD vs. Agile);

Integração com ferramentas de automação (ex.: depuradores inteligentes para validação em tempo real);

Expansão para áreas emergentes (ex.: quantum computing, onde Debugging requer novas abordagens).

Conclusão: O DAAD emerge como uma evolução pedagógica de _framework_s tradicionais, combinando rigor acadêmico com demandas industriais. Sua originalidade reside na ciclicidade entre criação e validação – um gap não endereçado por modelos concorrentes.

# Referências


[1] CAMBRIDGE MATHS. **Cambridge Maths.**, [s.d.][s.d.]. Disponível em: <https://www.cambridgemaths.org/>. Acesso em: 12 jul. 2025

[2] AI INDEX STEERING COMMITTEE. **2025 AI Index Report**. Stanford, CA: Stanford Institute for Human-Centered Artificial Intelligence, 2025. Disponível em: <https://hai.stanford.edu/ai-index/2025-ai-index-report>. Acesso em: 10 jul. 2025.

[3] CHUI, M. et al. **The state of AI in 2022—and a half decade in review**. [s.l.] McKinsey & Company, 2022. Disponível em: <https://www.mckinsey.com/~/media/mckinsey/business%20functions/quantumblack/our%20insights/the%20state%20of%20ai%20in%202022%20and%20a%20half%20decade%20in%20review/the-state-of-ai-in-2022-and-a-half-decade-in-review.pdf>. Acesso em: 10 jul. 2025.

[4] BRASIL. **Síntese de área: Ciência da Computação (Bacharelado/Licenciatura)**. Brasília, DF: Ministério da Educação, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep), Diretoria de Avaliação da Educação Superior (Daes), 2021. Disponível em: <https://download.inep.gov.br/educacao_superior/enade/relatorio_sintese/2021/Enade_2021_Relatorios_Sintese_Area_Ciencia_Computacao.pdf>. Acesso em: 10 jul. 2025.

[5] BRASIL. **Relatório Síntese de Área: Engenharia de Computação**. Brasília, DF: Ministério da Educação, Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep), Diretoria de Avaliação da Educação Superior (Daes), 2023. Disponível em: <https://download.inep.gov.br/educacao_superior/enade/relatorio_sintese/2023/engenharia_de_computacao.pdf>. Acesso em: 10 jul. 2025.

[6] LITHNER, J. A research framework for algorithmic and creative reasoning. **Educational Studies in Mathematics**, v. 67, n. 3, p. 255–276, 2008.

[7] HARISMAN, Y. et al. Exploring Students' Mathematical Reasoning Behavior. **Education Sciences**, v. 13, 2023.

[8] HURRELL, D. P. Conceptual knowledge OR Procedural knowledge OR Conceptual knowledge AND Procedural knowledge: why the conjunction is important for teachers. **Australian Journal of Teacher Education**, v. 46, n. 2, p. art. 4, 2021.

[9] NATIONAL COUNCIL OF TEACHERS OF MATHEMATICS – NCTM. **Principles and Standards for School Mathematics**. Reston, VA: NCTM, 2000.

[10] HIEBERT, J.; LEFEVRE, P. Conceptual and procedural knowledge in mathematics: An introductory analysis. Em: HIEBERT, J. (Ed.). **Conceptual and procedural knowledge: The case of mathematics**. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986. p. 1–27.

[11] KALDEWAIJ, A. **Programming: The Derivation of Algorithms**. [s.l.] Prentice Hall, 1990.

[12] GIBBONS, J. **Algorithm Design with Haskell**. University of Oxford, 2020. Acesso em: 10 jul. 2025

[13] BIRD, R.; DE MOOR, O. **Algebra of Programming**. [s.l.] Prentice Hall, 1997.

[14] GÜNDOĞDU, F. et al. Exploring mathematical reasoning skills. **ScienceDirect**, 2023.

[15] KOSMYNA, N. et al. **Your Brain on ChatGPT: Accumulation of Cognitive Debt when Using an AI Assistant for Essay Writing Task.**, 2025. Disponível em: <https://arxiv.org/abs/2506.08872>

[16] SILVA QUINTO, W. A. et al. Explorando o impacto da Inteligência Artificial na formação do pensamento crítico entre acadêmicos de T.I. na Região Norte do Brasil. **Caderno Pedagógico**, v. 22, n. 7, 2025.

[17] HSU, T.-C.; CHANG, Y.-S.; CHEN, S.-Y. Teaching AI with games: the impact of generative AI drawing on computational thinking skills. **Education and Information Technologies**, 2025.

[18] TSAI, C.-Y.; YANG, Y.-F. The impact of unplugged activities on developing computational thinking skills in elementary school students. **Journal of Educational Technology & Society**, v. 22, n. 3, p. 77–89, 2019.

[19] POLAT, E.; YILMAZ, R. M. Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students. **Education and Information Technologies**, v. 27, p. 9145–9179, 2022.

[20] BRACKMANN, C. P. et al. Unplugged versus plugged-in: examining basic programming achievement and computational thinking of 6th-grade students. **Journal of Computer Science Education**, 2022.

[21] GIBBONS, J. Functional Algorithm Design, Part 0. **SIGPLAN Blog**, 2020.

[22] LEHMANN, T. H. How current perspectives on algorithmic thinking can be applied to students' engagement in algorithmatizing tasks. **Mathematics Education Research Journal**, v. 36, n. 3, p. 609–643, 2024.

[23] WING, J. M. Computational thinking. **Communications of the ACM**, v. 49, n. 3, p. 33–35, mar. 2006.

[24] LEHMANN, T. H. Using algorithmic thinking to design algorithms: The case of critical path analysis. **The Journal of Mathematical Behavior**, v. 71, p. 101079, 2023.

[25] KONG, S.-C. et al. **Computational Thinking Education**. Singapore: Springer, 2019.

[26] SBC (SOCIEDADE BRASILEIRA DE COMPUTAÇÃO). **Referenciais de Formação em Computação: Educação Básica**. Porto AlegreSBC, 2017.

[27] PAPERT, S. **Logo: Computadores e Educação**. São Paulo: Brasiliense, 1985.

[28] HOARE, C. A. R. An Axiomatic Basis for Computer Programming. **Communications of the ACM**, v. 12, n. 10, p. 576–580, 1969.

[29] LEVESON, N. G. **Engineering a Safer World: Systems Thinking Applied to Safety**. Cambridge, MA: MIT Press, 2012.

[30] RIBEIRO, L. et al. Entendendo o Pensamento Computacional: além da programação. **Revista Brasileira de Informática na Educação**, v. 25, n. 3, p. 45–62, 2017.

[31] KONG, S. et al. **Pensamento Computacional na Educação: perspectivas internacionais**. São Paulo: Penso, 2020.

[32] MEDEIROS, W. M. **Pensamento Computacional ou Programação? Uma análise de práticas pedagógicas com Scratch**. Dissertação (Mestrado em Educação)—Uberlândia: UFU, 2024.

[33] SCRATCH FOUNDATION. **Scratch.**, [s.d.][s.d.]. Disponível em: <https://scratch.mit.edu/>. Acesso em: 7 jul. 2025

[34] HORA, N. DA. **O ensino do pensamento computacional no Brasil na era digital**. Futura, 9 fev. 2022. Disponível em: <https://futura.frm.org.br/conteudo/professores/artigo/o-ensino-do-pensamento-computacional-no-brasil-na-era-digital>. Acesso em: 9 jul. 2025

[35] FELIPUSSI, A. L.; PADUA, C. C. S. **Relato de aulas com robô programável e Pensamento Computacional**. Anais do 12º Congresso Brasileiro de Informática na Educação. **Anais**...Recife: SBC, 2023.

[36] FUTURA, C. P. **O ensino do pensamento computacional no Brasil na era digital**. São Paulo: Fundação Roberto Marinho, 2023.

[37] SAIDIN, N. D. et al. Benefits and challenges of applying computational thinking in education. **International Journal of Information and Education Technology**, v. 11, n. 5, p. 248–254, 2021.

[38] COMPUTER SCIENCE TEACHERS ASSOCIATION. **Computational Thinking: A Definition for K-12**., 2011. Disponível em: <https://csteachers.org/teaching-computational-thinking-in-early-elementary/>. Acesso em: 7 jul. 2025

[39] BONWELL, C. C.; EISON, J. A. **Active Learning: Creating Excitement in the Classroom**. Washington, D.C.: George Washington University, School of Education; Human Development, 1991.

[40] BRENNAN, K.; RESNICK, M. **New frameworks for studying and assessing the development of computational thinking**. Proceedings of the 2012 Annual Meeting of the American Educational Research Association. **Anais**...Vancouver, Canada: 2012.

[41] ARA'UJO, A. L. S. DE O.; ANDRADE, W. L. DE; GUERRERO, D. D. S. **Um mapeamento sistem'atico sobre a avaliaç ao do pensamento computacional no Brasil**. Anais do V Congresso Brasileiro de Inform'atica na Educaç ao (CBIE). **Anais**...2016.

[42] WEINTROP, D. et al. Defining computational thinking for mathematics and science classrooms. **Journal of Science Education and Technology**, v. 25, n. 1, p. 127–147, 2016.

# 8 Apêndice 1 - Atividades Unplugged

Estas são as sugestões para atividades a serem realizadas no primeiro dia de aula, logo após a apresentação da disciplina. O professor deve escolher as atividades que são adequadas ao seu perfil de aula e as características dos estudantes.

## 8.1 Fazer um bolo e Fazer um sanduíche de presunto e queijo

Nos dois casos, o professor deve distribuir uma página com pautas contendo a receita do bolo, ou do sanduíche. Divida a sala em grupos e forneça uma receita para cada aluno do grupo. A tarefa consiste em escrever, o mais detalhadamente todas as instruções necessárias para que alguém, que nunca fez um sanduíche, ou um bolo, possa fazer o sanduíche ou o bolo.

O professor deve enfatizar, todas as tarefas possíveis.

**Material necessário**: lápis e papel.

**Entrega**: a entrega será a lista de tarefas, escritas a mão.

**Avaliação**: o professor deve criar uma tabela com 10 notas no quadro, embaralhar as listas, para remover a identificação dos grupos mantendo cada lista anônima e ler cada lista de tarefas destacando os pontos que não foram indicados. Por exemplo: é impossível colocar o queijo no pão sem pegar a fatia de queijo antes. Caberá ao professor destacar as instruções faltantes. Uma vez que a lista tenha sido lida, o professor irá pedir que os alunos levantem a mão para a nota que eles acham que o grupo merece começando em 10 e marcar a nota que for mais votada. Separe a lista que tiver a maior nota ela será útil para explicar fluxogramas.

**Objetivo**: entender o que é dividir um problema em problemas menores e enfatizar a necessidade de detalhamento.

## 8.2 Programar o Professor

O professor é um automato que só sabe ler e identificar símbolos gráficos e usar o teclado. Além disso o professor tem acesso exclusivo aos seus segredos, como se fosse uma memória interna. Esta tarefa é para ser realizada por todos os alunos da sala que se voluntariem, sem escolha anterior, para determinar as tarefas que o professor deve realizar para passar um e-mail para si mesmo, ou outra tarefa simples.

Antes de começar Certifique-se que seja possível operar o seu sistema operacional apenas com as teclas `tab` e `enter` para conseguir realizar a tarefa que será proposta.

Em sala, peça a um aluno para anotar todos as instruções que foram realizados com sucesso. Guarde esta lista de instruções para a primeira aula de fluxogramas.

Para começar a atividade explique os limites da capacidade do professor, compartilhe o desktop do seu computador, sem nenhum aplicativo aberto. Informe que aplicativo, ou site, será necessário para completar a tarefa e pare com as mãos no teclado.

O professor deve ter apenas duas reações: ou faz o que os alunos pediram ou fica parado imóvel.

Esta é uma tarefa lúdica que deve provocar interesse e engajamento.

**Material necessário**: lápis e papel.

**Entrega**: lista de tarefas executadas criada pelos alunos.

**Avaliação**: esta atividade não requer qualquer tipo de avaliação.

**Objetivo**: mostrar os limites de entrada e saída de dados, os limites dos sistemas computacionais, a divisão de problemas em problemas menores.

## 8.3 Importante

Estas tarefas são apenas sugestões. O importante é que o professor seja capaz de fazer atividades lúdicas que envolvam toda a turma para a criação de listas de instruções para resolver tarefas simples, relacionadas com atividades cotidianas para criar os constructos cognitivos necessários ao entendimento da divisão de problemas grandes em problemas menores.