Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 03/Dec/2024; 1st round notif.: 19/Mar/2025; New version: 02/May/2025; 2nd round notif.: 13/Jul/2025
Camera ready: 21/Sep/2025; Edition review: 13/Oct/2025; Available online: 27/Oct/2025; Published: 27/Oct/2025

# Dificuldades no aprendizado de programação: Um exame baseado em mapeamento sistemático da literatura e interpretação analítico-comportamental

## *Title: Difficulties in Learning Programming: An Examination Based on a Systematic Literature Mapping and a Behavioral-Analytical Interpretation*

## *Título: Dificultades en el Aprendizaje de la Programación: Un Examen Basado en un Mapeo Sistemático de la Literatura y una Interpretación Analítico-Conductual*

*Marcelo H. Oliveira Henklain*
*Universidade Federal de Roraima*
*ORCID: 0000-0001-9884-8592*
*marcelo.henklain@ufrr.br*

*Leandro Silva Galvão de Carvalho*
*Universidade Federal do Amazonas*
*ORCID: 0000-0003-2970-2084*
*galvao@icomp.ufam.edu.br*

*Eduardo Luzeiro Feitosa*
*Universidade Federal do Amazonas*
*ORCID: 0000-0001-6401-3992*
*efeitosa@icomp.ufam.edu.br*

## Resumo

*Aprender a programar é uma demanda para muitos cidadãos do século XXI. Contudo, dificuldades nesse aprendizado persistem desafiando educadores. Para colaborar com a mitigação desse problema, definimos dois objetivos para este estudo: (1) Identificar dificuldades de alunos no aprendizado inicial de programação a partir de um mapeamento sistemático da literatura (MSL); e (2) Propor variáveis, baseadas na teoria psicológica analítico-comportamental e passíveis de investigação por pesquisas futuras, associadas a essas dificuldades. Para o MSL, definimos uma string de busca que foi aplicada em três bases de dados, Web of Science, ACM Digital Library e IEEE Xplore. Encontramos 503 estudos. Após a aplicação dos critérios de inclusão e exclusão, restaram 11. Apenas quatro pesquisadores apresentaram mais de uma publicação. Esses estudos se concentram na última década e são, tipicamente, publicados em periódicos, têm natureza empírica e identificam dificuldades em programação a partir da percepção de estudantes ou professores, com destaque para características da linguagem e conceitos de programação, bem como comportamentos como resolver problemas, ler e interpretar código e identificar fonte de erros. Para o segundo objetivo, consideramos que a teoria analítico-comportamental interpreta fenômenos psicológicos como comportamentos e os analisa como sistema de interações entre ambiente antecedente, ações da pessoa e ambiente consequente. Assim, a partir dos resultados do MSL e da experiência dos autores, propusemos 9 variáveis relativas ao ambiente antecedente que são dificeis de aprendermos a discriminar e, por isso, favorecem dificuldades em programação, bem como 9 relativas ao ambiente consequente, relacionados ao contexto de ensino, que podem tornar o processo de aprendizado menos efetivo. Destacamos, ainda, alguns repertórios comportamentais importantes de desenvolver para o sucesso em tarefas de programação. Esperamos com este trabalho fomentar estudos de base analítico-comportamental que busquem compreender e intervir sobre dificuldades no aprendizado de programação.*

***Palavras-Chave:*** *Educação em Computação; Dificuldades para Aprender a Programar; Análise do Comportamento*

## Abstract

*Learning to program is a demand for many 21st-century citizens. However, difficulties in this learning process persist, challenging educators. To help mitigate this problem, we defined two objectives for this study: (1) To identify students' difficulties in the initial learning of programming based on a systematic literature mapping (SLM); and (2) To propose variables, based on behavioral-analytical psychological theory and open to future research, associated with these difficulties. For the SLM, we defined a search string that was applied in three databases: Web of Science, ACM Digital Library, and IEEE Xplore. We found 503 studies. After applying inclusion and exclusion criteria, 11 remained. Only four researchers had more than one publication. These studies are concentrated in the last decade*

*Cite as: Henklain, M. H. O., Carvalho, L. S. G. & Feitosa, E. L. (2025). Dificuldades no aprendizado de programação: Um exame baseado em mapeamento sistemático da literatura e interpretação analítico-comportamental. Revista Brasileira de Informática na Educação, 33, 1484-1521. https://doi.org/10.5753/rbie.2025.5272*

*and are typically published in journals, are empirical in nature, and identify programming difficulties based on the perceptions of students or teachers, with emphasis on language features and programming concepts, as well as behaviors such as problem-solving, reading and interpreting code, and identifying sources of errors. For the second objective, we considered that behavioral-analytical theory interprets psychological phenomena as behaviors and analyzes them as a system of interactions between the antecedent environment, the person's actions, and the consequent environment. Thus, based on the SLM results and on our experience, we proposed 9 variables related to the antecedent environment that are difficult for us to discriminate and, therefore, contribute to difficulties in programming, as well as 9 variables related to the consequent environment, associated with the teaching context, that may make the learning process less effective. We also highlight some behavioral repertoires that are important to develop for success in programming tasks. With this work, we hope to foster behavior-analytic studies aimed at understanding and addressing difficulties in learning programming.*
***Keywords:*** *Computer Education; Difficulties in Learning to Program; Behavior Analysis*

## Resumen

*Aprender a programar es una demanda para muchos ciudadanos del siglo XXI. Sin embargo, las dificultades en este aprendizaje persisten, desafiando educadores. Para colaborar en la mitigación de este problema, definimos dos objetivos: (1) Identificar dificultades de los estudiantes en el aprendizaje inicial de programación a partir de un mapeo sistemático de la literatura (MSL); y (2) Proponer variables, basadas en la teoría psicológica analítico-conductual y susceptibles de ser investigadas en futuras investigaciones, asociadas a estas dificultades. Para el MSL, definimos una cadena de búsqueda aplicada en tres bases de datos: Web of Science, ACM Digital Library e IEEE Xplore. Encontramos 503 estudios. Tras aplicar los criterios de inclusión y exclusión, quedaron 11. Solo cuatro investigadores presentaron más de una publicación. Estos estudios se concentran en la última década y son, típicamente, publicados en revistas, tienen naturaleza empírica e identifican dificultades en programación a partir de la percepción de estudiantes o profesores, destacando características del lenguaje y conceptos de programación, así como comportamientos como resolver problemas, leer e interpretar código e identificar la fuente de errores. Para el segundo objetivo, consideramos que la teoría analítico-conductual interpreta los fenómenos psicológicos como comportamientos y los analiza como un sistema de interacciones entre el ambiente antecedente, las acciones de la persona y el ambiente consecuente. Así, a partir de los resultados del MSL y en nuestra experiencia, propusimos 9 variables relativas al ambiente antecedente que son difíciles de discriminar y favorecen dificultades en programación, así como 9 relativas al ambiente consecuente, relacionadas con el contexto de enseñanza, que pueden hacer que el proceso de aprendizaje sea menos efectivo. También destacamos algunos repertorios conductuales importantes de desarrollar para tener éxito en tareas de programación. Esperamos que este trabajo fomente estudios de base analítico-conductual que busquen comprender e intervenir en las dificultades del aprendizaje de la programación.*
***Palabras clave:*** *Educación en Computación; Dificultades para Aprender a Programar; Análisis del Comportamiento*

# 1 Introdução

Programar computadores, de um ponto de vista analítico-comportamental, é um comportamento humano complexo, mas isso não significa que aprendê-lo precise ser uma experiência aversiva ou frustrante (Lazzari, 2013; Fontoura-Júnior et al., 2023). Não obstante, as taxas de retenção e evasão em disciplinas do ensino superior de introdução a programação persistem elevadas no Brasil e no mundo (Pereira et al., 2019; Castro & Tedesco, 2020). Devemos perguntar, então, quais são as dificuldades relacionadas ao ensino de programação que justificam esse cenário e como essas dificuldades podem ser interpretadas pela teoria analítico-comportamental de modo a orientar mais pesquisas sobre essa temática?

Essas são questões relevantes de serem respondidas. Estudo recente projetou que, no período de 2021 a 2025, a indústria de tecnologia da informação e comunicação precisou contratar em média 159 mil profissionais de informática por ano no Brasil. Contudo, são formados apenas 53 mil, ou seja, tivemos um déficit de 106 mil profissionais a cada ano nesse período (Brasscom, 2021). Para muitos desses profissionais, a programação de computadores é uma habilidade crucial. Mesmo em um cenário pessimista de perda de empregos em razão de crise econômica ou evolução de ferramentas de Inteligência Artificial (IA), capazes de substituir postos de trabalho, é preciso considerar que precisaremos de profissionais ainda mais capacitados para a tarefa de programação de modo a manter e evoluir IAs e outras tecnologias (Yusoff et al., 2020).

Além disso, em um mundo marcado pela digitalização de todos os processos, necessitamos de uma sociedade minimamente letrada para interpretar códigos e, caso não seja para criar programas, adaptá-los para as suas necessidades, nas mais diversas áreas de atuação profissional e esferas da vida (Unesco, 2014; Satub & Chothia, 2022). Ademais, para desenvolvermos o pensamento computacional em todos, visto que se trata de habilidade necessária para o cidadão do século XXI, também precisamos do ensino de introdução à programação desde a educação básica. Programar é um dos principais meios identificados de promoção desse tipo de pensamento que, hoje, já compõe, inclusive, currículos da educação básica em diferentes países do mundo, incluindo o Brasil (Wing, 2006; BNCC, 2018; Santana et al., 2021).

É, portanto, evidente a demanda pela superação de obstáculos em relação ao aprendizado de programação. Também é preciso que as pessoas que desejem trabalhar com programação aprendam a "programar computadores com conforto e sentido", isto é, sem que a tarefa de programar seja exaustiva e sem que a pessoa dependa excessivamente de auxílio para entender o que está fazendo ou como deve proceder para programar corretamente (Lazzari, 2013). Nesse contexto, descobrir, especificamente, quais são as dificuldades dos alunos é importante porque professores precisam desse conhecimento para planejar o ensino. Afinal, sabendo os obstáculos que poderão enfrentar, estarão melhor preparados para superá-los ou atenuá-los (Qian & Lehman, 2022). As dificuldades também nos indicam aprendizagens que deveriam ter sido garantidas previamente, permitindo um conhecimento mais preciso do que, efetivamente, é requerido de uma pessoa na tarefa de programação, o que permite ao professor aperfeiçoar recursos avaliativos e de ensino (Fontoura-Júnior et al., 2023). O conhecimento das dificuldades é relevante também para desenvolvedores de linguagens de programação e ambientes de desenvolvimento, pois conhecendo-as é possível, por exemplo, melhorar mensagens de erro.

No que diz respeito ao mapeamento de dificuldades no aprendizado de programação, dispomos de muitos estudos (e.g., Qian & Lehman, 2017; Yusoff et al., 2020; Araújo et al., 2021; Alasmari et al., 2024). Não obstante, poucos se dedicam a identificar motivos para as dificuldades e tipos de dificuldades mais complexas, existindo uma tendência a enfatizar problemas relativos ao aprendizado de sintaxe de uma linguagem (Satub & Chothia, 2022). Notamos também que, geralmente, na literatura de informática na educação e educação em computação, poucos estudos examinam os processos de ensinar e aprender ou dificuldades neles à luz da teoria analítico-

comportamental. Em uma consulta rápida à Biblioteca Digital da SBC, em abril de 2025, com a *string* "análise do comportamento" OR "analítico-comportamental" OR "behaviorismo radical" OR "behaviorismo" e o critério de que o descritor poderia aparecer em qualquer lugar do artigo, podendo ter sido publicado em anais ou periódicos, em qualquer data, encontramos apenas 24 estudos. Desses, apenas 7 eram analítico-comportamentais. Nenhum deles examinava os processos de ensinar e aprender programação. Esse dado é preocupante porque a Análise do Comportamento é uma teoria científica que estuda e intervém sobre fenômenos psicológicos, tendo sido bem-sucedida quando aplicada a diversos contextos, com destaque para o campo educacional (Cianca et al., 2020). A propósito, a Análise do Comportamento Aplicada à Educação tem sido efetiva até mesmo nos casos mais desafiadores, envolvendo pessoas com desenvolvimento atípico e transtornos de aprendizagem (Heward et al., 2022). Por esse motivo, é razoável trazê-la para o debate sobre dificuldades no ensino de programação.

Neste estudo, o que pretendemos foi (1) identificar dificuldades de alunos no aprendizado inicial de programação a partir de um mapeamento sistemático da literatura (MSL) e (2) propor variáveis, baseadas na teoria psicológica analítico-comportamental e passíveis de investigação por pesquisas futuras, associadas a essas dificuldades. Tais variáveis são, conforme explicaremos, propriedades de estímulos associados à programação ou do repertório comportamental requerido para que se consiga programar. Complementarmente, apresentamos uma atualização da análise do comportamento de "programar computadores com conforto e sentido", elaborada por Lazzari (2013) (ver Apêndice 1), que busca descrever os comportamentos mais específicos que o constituem e que podem ser adotados como objetivos de aprendizagem (tecnicamente denominados de "comportamentos-objetivo") em disciplinas de introdução à programação. Esse exame pode ser útil para indicar quais aprendizagens devem ser garantidas, quando almejamos que alguém saiba programar, e, portanto, também ajuda a entender dificuldades com a programação. Conforme exposto, essas 2 contribuições, proposição de variáveis e análise do comportamento de programar computadores, serão feitas com base na ciência da Análise do Comportamento (AC), que será apresentada neste trabalho de forma introdutória.

O nosso MSL contribui com a literatura porque (1) incluiu a Web of Science, base de dados internacional de prestígio, mas pouco adotada nas revisões de literatura existentes sobre dificuldades com a programação, (2) incluiu estudos com diferentes públicos de alunos aprendendo programação (crianças, jovens, adultos, cursando ou não graduação em computação), permitindo uma visão mais ampla dos tipos de dificuldades dos iniciantes, (3) enfatizou o exame de dificuldades básicas e complexas no aprendizado de programação e (4) disponibilizou a base de dados da pesquisa, para que possa ser criticada e aperfeiçoada por estudos futuros. Com relação à proposição de variáveis associadas a dificuldades e a análise do comportamento de programação de computadores, ressaltamos que consiste em uma proposta dos autores a partir de nossa experiência com AC e educação em computação, bem como da leitura realizada dos estudos incluídos no MSL. Não temos a pretensão de que esta proposta seja definitiva. Por ser algo preliminar, não adotamos procedimento sistemático na confecção desses dois produtos. A expectativa é que ambos ilustrem possibilidades de aplicação da AC ao exame de dificuldades no aprendizado de programação e que sirva como subsídio para pesquisas futuras, em relação as quais apresentaremos propostas concretas.

Organizamos este trabalho em 6 seções. Na fundamentação teórica, caracterizamos a AC enquanto disciplina científica e apresentamos uma breve nota sobre aspectos a considerar em um exame analítico-comportamental acerca do comportamento de programar computadores. Na seção de trabalhos relacionados, revisamos a literatura e destacamos as novidades deste estudo. No método, explicitamos os critérios para a condução do MSL e descrevemos como propusemos variáveis e atualizamos o exame de Lazzari (2013). Na seção de resultados e discussão,

descrevemos os nossos achados e, na conclusão, apresentamos a nossa resposta ao problema de pesquisa, destacando implicações educacionais, limitações e proposta para trabalhos futuros.

# 2 Fundamentação teórica

## 2.1 Perspectiva psicológica adotada

Neste estudo, interpretaremos os fenômenos psicológicos envolvidos na programação de computadores à luz da Análise do Comportamento (AC). Essa ciência pode ser caracterizada (Carvalho-Neto, 2002; Todorov, 2007; Carrara & Strapasson, 2014; Botomé, 2015; Zilio & Neves-Filho, 2018) como analítica, pois decompõe fenômenos em seus processos constituintes para entendê-los melhor e, assim, alcançar a compreensão do todo. Possui natureza experimental e, por isso, requer o teste sistemático de relações entre variáveis independentes sobre dependentes, tanto em ambientes controlados, como naturais, aceitando, nesse cenário, as limitações inerentes de controle experimental. Notabiliza-se por eleger o comportamento como seu objeto de estudo e não como uma medida indireta de outros processos. Entende o comportamento como fenômeno natural, em uma perspectiva monista materialista, que consiste em um complexo sistema de interações entre ambiente (histórico ou imediato, interno ou externo, físico ou social, que antecede e sucede o que o organismo faz) e ações que um organismo como um todo pode apresentar, a partir de seu aparato biológico. A AC se dedica a teorizar sobre o comportamento a partir de dados, em uma lógica indutiva e buscando encontrar meios de descrever da forma mais precisa e parcimoniosa possível correlações observadas entre (1) eventos ambientais antecedentes, (2) ações do organismo e (3) eventos ambientais consequentes. Esses três tipos de eventos constituem a unidade de análise do fenômeno comportamental, sendo denominada de tríplice contingência.

Se consideradas múltiplas ocorrências de um comportamento, é possível notar que cada um dos seus três componentes (antecedente, ação e consequente) pode ocorrer de formas ligeiramente distintas, mas mantendo a função. Por exemplo, uma ação de digitar pode ocorrer de diferentes formas (com uma só mão, com as duas), mas sempre com a mesma função de produzir palavras na tela do computador. Estímulos, por sua vez, antecedentes ou consequentes, podem variar em suas propriedades de duração, intensidade etc. Por isso, analistas do comportamento usam a "expressão" classe de ações ou de antecedentes e consequentes. A ideia é que diferentes formas de apresentação compõem uma mesma classe, quando compartilham uma função.

Sobre o termo comportamento, importa esclarecer que, na tradição fisiológica, ele pode aparecer associado apenas à dimensão observável de movimentos ou atividades de partes do organismo, podendo envolver desde o funcionamento de um órgão até o recrutamento da musculatura esquelética para a promoção do deslocamento do corpo (Carvalho-Neto, 2002). Contudo, o sentido dado ao termo comportamento na teoria analítico-comportamental é distinto, por isso a ênfase nas expressões "organismo como um todo" e "relação entre ambiente e ações". Trata-se, portanto, de considerar como pessoas interagem com o seu mundo, no lugar de um exame da ação em si mesma, por exemplo, em termos de sua forma de apresentação.

Processos de interesse para a educação em computação, como "raciocinar logicamente sobre a relação existente entre componentes de um problema", "resolver problemas considerando requisitos do problema" e "programar computadores com conforto e sentido" são interpretados pela AC como comportamentos. Neles, a dimensão da ação é representada por um verbo no infinitivo e a dimensão do ambiente é sintetizada em um complemento, como "... logicamente sobre a relação existente entre componentes de um problema". Para a AC, não há problema algum que processos envolvidos no comportamento, como é o caso do raciocínio lógico, ocorram de modo que só a pessoa que se comporta possa observá-los (Todorov, 2007). A publicidade ou

externalidade do fenômeno não é o que importa para que se possa realizar um exame analítico-comportamental (Zilio & Neves-Filho, 2018).

O foco está, repetimos, no exame das relações de intercâmbio entre ambiente e ações. Algumas perguntas podem ilustrar a que relações estamos nos referindo: em que contexto a pessoa agiu? como esse contexto impactou esse fazer, favorecendo-o ou não? contextos ligeiramente distintos continuariam produzindo o mesmo efeito? contextos sem qualquer relação física entre si podem ter o mesmo efeito? que mudanças no mundo esse fazer produziu após ter sido emitido? como tais alterações no mundo impactaram a probabilidade de recorrência desse fazer, sua frequência de emissão e força (no sentido de persistência ao longo do tempo e diante de novas circunstâncias)? em que medida essas mudanças no mundo repercutiram sobre o significado que o contexto atual passará a ter em futuras interações? etc. Portanto, da definição de comportamento apresentada, deve ficar explícito que não há limitação a priori sobre a sua aplicação a processos e fenômenos psicológicos, tradicionalmente, denominados como cognitivos ou emocionais, ou aqueles que são chamados de complexos, criativos e subjetivos.

Contudo, é preciso lembrar que as explicações para o comportamento segundo a AC podem diferir do que aquelas do senso comum. No dia a dia, é usual que se observe uma pessoa agir e, então, que se infira ou crie uma entidade que habita dentro dessa pessoa, para explicar a sua ação. Por exemplo, se uma criança, tipicamente, faz muitas perguntas (ação), isso é explicado em função da sua "curiosidade" (entidade interna). Essa entidade foi apenas inferida do que pode ser observado e, na prática, acrescenta pouco ao conhecimento científico sobre porque a criança pergunta tanto. O motivo é que a "curiosidade" é apenas uma repetição do que já sabemos, gerando uma explicação circular: sabe-se que a pessoa é curiosa porque faz muitas perguntas, e faz tantas perguntas porque é curiosa. Em um exame cuidadoso, vemos que a expressão "curiosidade", provavelmente, surge com a função de nomear um conjunto de comportamentos, que envolve perguntar sobre temas diversos, em muitas situações. De nome, passa, então, inadvertidamente, a adquirir o papel de entidade com capacidade de causar o comportamento. Na AC, explicar requer descrever as complexas relações históricas e presentes entre ambiente e ações da pessoa para fornecer explicações sobre por qual motivo a pessoa se comportou de determinado modo (Leão & Laurenti, 2009; Carrara & Strapasson, 2014). Cabe à fisiologia e neurociência e não a AC o papel de explicar como, biologicamente, esse comportamento foi viabilizado.

A interpretação de processos e fenômenos psicológicos como comportamentos e a estratégia analítica e experimental da AC, levaram à descoberta de múltiplos princípios de aprendizagem que ajudam a compreender e manejar comportamentos de modo mais efetivo (Moreira & Medeiros, 2018). A aplicação dessas descobertas permitiu o desenvolvimento de uma ciência e tecnologia com um relevante arcabouço de contribuições para a humanidade. Tal impacto social abarca desde o tratamento de pessoas em sofrimento mental e com desenvolvimento atípico, com destaque para o transtorno do espectro autista, até intervenções para a promoção de comportamentos pró-ambientais, prevenção de acidentes no trabalho, manejo de condutas em sala de aula, desenvolvimento de habilidades de informática, adesão a tratamentos de saúde, ampliação do potencial criativo, prevenção de evasão em cursos, redução de comportamentos violentos, atenuação na propagação de *fake news*, mudança de comportamento alimentar, entre outros exemplos. Até o momento, além do que citamos, existe um rol proposto de 350 domínios, que ilustram como a AC pode ser útil para nos ajudar a manejar de modo mais efetivo diferentes classes de comportamento humano (Heward et al., 2022).

Tendo em vista o sucesso da AC em outros âmbitos, com destaque para a educação (Cianca et al., 2020), acreditamos que ela também pode trazer contribuições ao campo da informática na educação e da educação em computação. Um tema importante nessa subárea da computação diz respeito às dificuldades no aprendizado de programação. Identificamos que ele já começou a ser investigado por analistas do comportamento (Lazzari, 2013) e pretendemos, neste estudo, como

parte do nosso segundo objetivo, ampliar esse exame, bem como incentivar que continue e se fortaleça a colaboração entre analistas do comportamento e educadores.

## 2.2 Breve interpretação analítico-comportamental sobre o comportamento de programar computadores

Interpretar à luz da AC em que consiste programar envolve examinar esse fenômeno à luz da tríplice contingência, isto é, como processo comportamental, composto por classes de eventos ambientais antecedentes, classes de ações e classes de eventos ambientais consequentes interrelacionadas (Lazzari, 2013; Botomé, 2015). Classes de comportamentos podem manter entre si uma relação de encadeamento, na qual o ambiente consequente produzido por uma classe de ação é ambiente antecedente para a ocorrência de outra classe de ação. É possível existir também uma relação de composição, em que várias classes de comportamento mais específicas são pré-requisitos para que outra classe, mais abrangente, possa ocorrer.

Neste estudo, com base em Lazzari (2013), consideramos a classe geral de comportamentos "programar computadores com conforto e sentido" como composta por sete classes de comportamentos mais específicas, encadeadas entre si de modo que, diante de um problema, seja possível encontrar uma solução algorítmica e construir um programa para implementá-la, resolvendo o problema. Essas classes são as seguintes, conforme a ordem de seu encadeamento: (1) Avaliar argumentos de acordo com regras lógicas; (2) Caracterizar funcionamento de computadores; (3) Resolver problemas; (4) Construir algoritmos; (5) Formalizar algoritmos; (6) Escrever programas de computador; (7) Avaliar programas de computador. Cada uma das sete classes de comportamentos mais específicas pode, por sua vez, ser decompostas em classes de comportamento ainda mais específicas, sendo que o grau de especificidade dessa descrição depende de quanta precisão é necessária para um determinado fim, como o ensino desses comportamentos. Para um processo de ensino pode ser útil chegar a descrições de comportamentos tão simples que o professor poderia começar por eles, pois, provavelmente, seriam aprendidos com menos esforço, permitindo uma evolução gradual do estudante, indo do aprendizado de comportamentos mais simples para os mais complexos (Cortegoso & Coser, 2023). Esse processo de análise e decomposição de comportamento é útil para o professor identificar claramente o que precisa ser ensinado e como avaliar se cada aprendizagem relevante foi alcançada e em que grau. Esclarecemos que, ao descrever comportamentos, o verbo no infinitivo indica a ação e os complementos ao verbo podem conter informações sobre eventos ambientais antecedentes e/ou consequentes. Usar apenas verbo e complemento gera uma descrição mais genérica, mas que é útil por ser mais fácil de comunicar, do que descrever o comportamento em formato de tabela com uma descrição pormenorizada da tríplice contingência.

Uma dúvida comum em relação a AC diz respeito a como interpretar processos cognitivos tipicamente associados à programação de computadores, como "raciocínio lógico" e "pensamento". A solução consiste em tratar esses termos e construtos cognitivos como respostas verbais do cientista ou do educador em computação e buscar, então, os seus significados entre os determinantes ambientais dessas respostas verbais (Sério, 2005). Precisamos, portanto, avaliar sob que condições termos como "raciocínio lógico" e "pensamento" são usados para falar sobre programação de computadores ou, ainda, que tipos de comportamentos costumam ser entendidos como "raciocínio lógico" ou "pensamento". De acordo com Bandini e Delage (2012), o raciocínio é, tipicamente, usado no contexto de ser um tipo de pensamento, motivo pelo qual vamos ilustrar como examiná-lo. Esse exame será útil para a interpretação que faremos nas últimas seções do estudo sobre as dificuldades no aprendizado de programação.

O termo pensamento, segundo Bandini e Delage (2012), parece ser usado em diferentes contextos, com destaque para falar de: (1) comportamentos específicos que pessoas podem emitir, como nos casos de "pensar matematicamente"; (2) agir em relação a um estímulo específico, como

quando falamos que estamos “pensando sobre...”; (3) algum processo comportamental, como ter discriminado a forma correta de agir em uma situação, que é quando falamos que a pessoa “pensou ou julgou corretamente como agir”; (4) no contexto de uma ação fraca, pois o controle discriminativo existente é insuficiente para sinalizar a ação esperada, condição em que falamos “eu penso que devo agir assim” no lugar de “eu sei que devo agir assim”; e, principalmente, de (5) comportamentos precorrentes, no sentido do que a pessoa faz antes de produzir uma solução e que aumenta as chances de que tal solução ocorra, sendo, portanto, uma parte fundamental do processo de resolução de problemas. Esse quinto uso, possivelmente, é o uso mais comum do termo pensamento e envolve, dentre outros, dois comportamentos mais básicos e precorrentes em relação à solução de problemas que são atentar e decidir.

O termo atenção é, tipicamente, usado para falar de ações que produzam como consequência melhorias no contato da pessoa com os estímulos antecedentes que precisam ser discriminados, para a adequada solução de um problema (Bandini & Delage, 2012). “Olhar para um estímulo”, “procurar propriedades de um estímulo”, “afastar estímulos potencialmente distratores” e “responder adequadamente a certas propriedades do estímulo” podem ser comportamentos mais específicos envolvidos no comportamento mais abrangente de “atentar para um estímulo de modo a aumentar as chances de solução do problema”. A análise de que pensar envolve atentar é útil para a educação porque orienta o professor sobre como, concretamente, ensinar estudantes de programação a pensar. Esse aprendizado pode começar por ensiná-los a atentar para os estímulos relevantes, como os requisitos do problema. Para a AC, o ambiente não tem um significado inequívoco, mudando em função da nossa história de aprendizagem em relação a ele. Por exemplo, a partir do ensino, partes do ambiente, antes sem significado específico, como a expressão “até que” em “o programa deve ser executado *até que* o usuário clique na tecla S”, presente no enunciado de um problema, podem adquirir funções de sinalização do caminho mais promissor para a resposta correta, como adotar um laço de repetição com uma condição de verificação sobre se a letra ‘S’ foi pressionada. Além de atentar, pensar para resolver um problema requer tomar decisões sobre o melhor curso de ação.

O termo decidir, por sua vez, também parece se referir a um comportamento precorrente que envolve “selecionar um curso de ação quando dois ou mais estão em conflito”, pois são, inicialmente, igualmente prováveis (Bandini & Delage, 2012). Para que uma decisão ocorra, é crítico “manipular o ambiente antecedente, em busca de um novo arranjo de estímulos que torne mais provável de ser emitido o curso de ação com maior potencial de sucesso para a solução do problema”. Dessa análise, o educador em computação pode notar que ensinar a pensar passa por ensinar seus alunos estratégias sobre como “buscar informações adicionais acerca do problema” ou sobre como “representar com diagramas os requisitos que precisam ser atendidos”, de modo a tornar mais evidente qual a melhor solução do problema. Avançando em nosso exame, podemos perguntar: o que é um problema e por que pensar sobre ele pode ser útil para solucioná-lo?

Segundo a AC, problema é uma situação na qual uma ação que produz a solução do problema não é possível de ser emitida, exceto se os comportamentos precorrentes forem apresentados e, por meio deles, um novo arranjo contextual for construído de tal modo que a ação que soluciona o problema possa, então, ser apresentada (Bandini & Delage, 2012). Note que, para falarmos que uma situação é problemática para uma pessoa, não é admissível que ela tenha sido diretamente treinada sobre como resolver tal situação. Os comportamentos precorrentes precisam ocorrer. Contudo, isso não significa dizer que essa pessoa não tenha tido algum treino em sua história de vida que possa lhe servir em face de um problema. Desconsiderando os casos de sorte, um problema só é passível de solução por alguém quando ela já teve algum treino relativo aos comportamentos específicos que compõem o comportamento mais geral, capaz de produzir a solução. No paradigma de recombinação de repertórios (Bandini & Delage, 2012), pesquisadores

estudam como comportamentos distintos, aprendidos na história da pessoa, podem, sob controle de uma situação-problema específica, se recombinar e, com isso, levar à solução do problema.

Um exemplo pode ajudar a entender como repertórios podem se recombinar. É o caso de primatas que aprendem, em uma seção de treino, a puxar para perto de si um cacho de bananas fora de sua gaiola com auxílio de uma ferramenta e, em outro treino, aprendem a conectar varetas para ganhar bananas. Em uma situação na qual bananas são colocadas fora da gaiola, mas a uma distância inalcançável, mesmo com a ajuda da ferramenta do Treino 1, se uma vareta que possa se conectar a essa ferramenta estiver disponível, a conexão de ambas pode aumentar o tamanho da ferramenta de captura de bananas, resolvendo o problema. Assim, quando entendemos a história de aprendizagem e as relações presentes entre eventos ambientais antecedentes, ações e eventos ambientais consequentes, ficam mais evidentes quais são os aspectos críticos no que chamamos de resolver problemas ou, de modo mais genérico, pensar e raciocinar. Também fica mais evidente para o educador em computação que é preciso precisão sobre o que significa programar computadores, para que consigamos ensinar esse comportamento de modo efetivo.

Para a AC, o nosso primeiro desafio é descobrir a história de aprendizagens e as intrincadas relações entre ambiente e ações do organismo, para que possamos entender melhor como são adquiridos comportamentos como pensar, raciocinar, resolver problemas e programar. Outro desafio crucial é o de mapear quais são as classes de comportamento que, se aprendidas, são suficientes para tornar uma pessoa apta a apresentar um comportamento que depende desses pré-requisitos. No contexto do ensino, cada classe de comportamento que precisa ser aprendida pode ser denominada de "comportamento-objetivo" (Kienen et al., 2021). Esse termo enfatiza que o processo de ensino deve iniciar pela proposição do que é esperado que o aluno aprenda, pois é a partir desse alvo que se podem construir instrumentos de aprendizagem e propor metodologias de ensino. Tal lógica supera a noção de que se ensinam "conteúdos", enfatiza o aluno como central no processo de ensino-aprendizagem e abre caminho para que se adotem, com alta efetividade, metodologias ativas, nas quais tarefas dos alunos, dentro e fora da sala de aula, serão orientadas para o desenvolvimento de comportamentos de valor, capazes contribuir com as suas vidas e atuações nas realidades sociais em que vivem (Cortegoso & Coser, 2023).

Neste estudo, coerentemente com nosso segundo objetivo, pretendemos atrair a atenção de pesquisadores de informática ne educação, educação em computação e tecnologias na educação em computação para que conduzam exames conceituais e pesquisas empíricas sobre aprendizado de programação de computadores à luz da AC. Para tanto, fornecemos nesta seção algumas explicações teóricas básicas. Na seção de resultados e discussão, apresentaremos tendências e lacunas no conhecimento científico a partir do Mapeamento Sistemático da Literatura (MSL) e discutiremos variáveis relevantes, de um ponto de vista analítico-comportamental, associadas às dificuldades com o aprendizado de programação. Por fim, retomaremos o trabalho de Lazzari (2013), relacionado à descoberta das classes de comportamento mais específicas, constituintes da classe geral "programar computadores com conforto e sentido". Assim, esperamos estimular o desenvolvimento de estudos sobre ensino de programação baseados em AC.

As referências que selecionamos para este texto tem uma função adicional de serem didáticas e mesclarem textos clássicos e atuais, de modo que pesquisadores interessados tenham elementos para avançar em estudos sobre AC. Destacamos que conceitos fundamentais de AC, como os de reforço e equivalência de estímulos, não foram suficientemente desenvolvidos ou mencionados em razão de limitação de espaço, devendo ser examinados em trabalhos futuros.

# 3 Trabalhos relacionados

Nesta seção, apresentamos estudos que nos permitiram identificar lacunas no conhecimento científico sobre aprendizado de programação de computadores e que, portanto, justificam este estudo. Silva et al. (2022), por exemplo, investigaram quais são os tópicos mais abordados na primeira disciplina de programação (CS1) que é ofertada em cursos de computação das universidades federais brasileiras. Foram obtidas 150 ementas de CS1 de 61 universidades federais dentre as 69, sendo a maior parte do Nordeste e, na sequência, do Sudeste. Nenhum tópico isoladamente esteve em todas as 150 ementas analisadas. Os tópicos com mais de 50% de ocorrência foram: variáveis, constantes e atribuições (92,7%); comandos condicionais (92%); comandos de repetição (90%); funções, modularização e subprogramas (86%); expressões aritméticas, lógicas e relacionais (85,3%); variáveis compostas homogêneas unidimensionais (83,3%); variáveis compostas homogêneas multidimensionais (80,7%); representações de algoritmos (76%); entrada e saída de dados (63,3%); e variáveis compostas heterogêneas (60%). Esses tópicos são compatíveis com o que tem sido encontrado em referências internacionais. Tais resultados sugerem que ainda não existe clareza sobre tópicos ou conteúdos a serem abordados pelo professor em sala de aula, sendo possível inferir que o conhecimento sobre comportamentos que precisam ser desenvolvidos em CS1, ou seja, o que se espera que os alunos aprendam a fazer em relação a esses tópicos e conteúdos, é ainda menor.

Com o MSL deste estudo, pretendemos conhecer mais sobre quais são as dificuldades mais comuns em programação e, a partir disso, complementar a análise do comportamento de "programar computadores com conforto e sentido" realizada por Lazzari (2013) e que é subsídio para a proposição de comportamentos-objetivo no contexto do ensino de programação. Nesse sentido, Silva et al. (2019) verificaram, por meio dos altos índices de reprovação nas disciplinas de programação, que de fato existe dificuldade para aprender programação, sendo a metodologia de ensino um aspecto importante para explicá-las. Por isso apresentamos neste estudo uma lista de variáveis associadas a dificuldades de aprendizado e de comportamentos-objetivo que professores podem considerar ao planejar e avaliar o ensino de programação.

No contexto das dificuldades no aprendizado de programação, Medeiros et al. (2020) buscaram por artigos brasileiros, publicados entre 2010 a 2016, sobre desafios de ensino e aprendizagem nas disciplinas de introdução à programação no ensino superior. Verificou-se que a falta de habilidades de interpretação de texto e déficits na formação em matemática, bem como infraestrutura física e de recursos humanos deficitária, são variáveis determinantes para as dificuldades no aprendizado de programação. Os pesquisadores concluíram, então, que o ensino superior brasileiro precisa de auxílio da comunidade de educação em computação. Atendendo ao chamado de Medeiros et al. (2020), decidimos conduzir o presente estudo, avançando em relação a esse trabalho ao buscarmos estudos em bases de dados internacionais.

Araújo et al. (2021), por sua vez, investigaram quais são os principais tipos de concepções equivocadas em estudantes de introdução à programação, que não cursam graduações em informática. Foram coletados dados de 39 alunos de Engenharia Mecânica em 2018 e 42 alunos de Engenharia de Materiais em 2019, sendo que essa turma de 2019 obteve a maior taxa de reprovação de uma série histórica de 20 semestres. Verificou-se que a turma de 2018 apresentou mais dificuldades com o módulo de vetores e strings, enquanto a turma de 2019 demonstrou maior dificuldade no módulo de laço por contagem. De modo geral, os tipos de concepções equivocadas mais frequentes se relacionaram a erros sintáticos e lógicos simples e erros específicos da linguagem Python. Neste trabalho, incluímos estudos envolvendo estudantes de computação e de outras áreas, desde crianças a adultos, o que nos permite ter uma visão ampla sobre o conhecimento que tem sido produzido acerca das dificuldades no aprendizado de programação.

Em síntese, observamos que as lacunas no conhecimento envolvem a falta de descrição de propriedades dos estímulos associados à programação que podem gerar dificuldades e do repertório que precisa ser desenvolvido para que se possa afirmar que uma pessoa sabe programar. Ademais, notamos uma ênfase na elaboração de recursos para ensinar e menos para auxílio ao processo de planejá-lo. Neste estudo, nos concentramos na identificação de dificuldades em programação e na proposição de variáveis associadas a essas dificuldades, o que pode repercutir positivamente sobre planejamento, implementação e avaliação do ensino.

# 4 Método

Conduzimos um Mapeamento Sistemático da Literatura (MSL), cuja finalidade é caracterizar a produção de conhecimento científico sobre determinado fenômeno e, assim, subsidiar a proposição de novos estudos (Wazlawick, 2021). Neste MSL adotamos as seguintes etapas, que são compatíveis com as recomendações de Petersen et al. (2015): (1) Formulação das perguntas de pesquisa a serem respondidas pelo MSL; (2) Definição dos critérios de inclusão e exclusão dos estudos; (3) Definição da string de busca; (4) Definição das fontes de busca dos estudos; (5) Coleta e análise de dados; e (6) Definição de estratégia para garantia da qualidade do estudo.

Após a execução do MSL, para estimular a condução de estudos analítico-comportamentais, executamos mais duas etapas: (7) Organização de lista de variáveis associadas às dificuldades no aprendizado de programação derivadas do MSL, bem como (8) Condução de análise dos comportamentos mais específicos constituintes do comportamento de "programar computadores com conforto e sentido", complementando o exame iniciado por Lazzari (2018).

## 4.1 Etapas do Mapeamento Sistemático da Literatura

### *4.1.1 Etapa 01 - Perguntas de pesquisa*

Este MSL responde a três questões de pesquisa sobre pesquisas relativas às dificuldades no aprendizado de introdução à programação, a saber: **PP01.** Quando, por quem e onde os estudos são tipicamente publicados? **PP02.** O que geralmente tem sido investigado e como? **PP03.** Quais são as dificuldades mais comuns no aprendizado de programação?

### *4.1.2 Etapa 02 - Critérios de inclusão e exclusão*

Incluímos estudos (1) completos, (2) publicados como artigos em periódicos ou eventos científicos ou capítulos de livro, (3) redigidos em português, espanhol ou inglês, (4) disponíveis na Internet via *open access* ou acesso institucional de um dos autores, bem como por meio do Portal de Periódicos da CAPES, e cujo objetivo (5) envolvesse o exame sobre quais são as dificuldades de estudantes no aprendizado de introdução à programação de computadores. Esses estudos poderiam ser exames teóricos, revisões de literatura ou investigações empíricas envolvendo, por exemplo, estudantes de programação e professores que lecionam disciplinas como introdução à programação (no inglês, *Computer Science* 1 ou CS1). Nesta pesquisa, em função do nosso escopo ir além do MSL, adotamos critério mais restritivo de não coletar estudos disponíveis em outras fontes como o ResearchGate.

Excluímos estudos que não atendiam a esses critérios e cujo objetivo fosse investigar (1) tema diverso da área de educação em computação, (2) avaliação ou aperfeiçoamento de mensagens de erro de compiladores / interpretadores, (3) proposta, intervenção ou caracterização de práticas para ensinar programação ou aperfeiçoar repertório, (4) ênfase exclusiva no exame da relação entre estados psicológicos e aprendizado de programação, (5) técnica de detecção, classificação ou mensuração de dificuldade de tarefa, (6) técnica de detecção, classificação ou

mensuração de erros, (7) técnica de verificação de plágio e, de modo geral, (8) estudos que não abordassem dificuldades no aprendizado inicial de programação, mas, por exemplo, em disciplinas mais avançadas de programação.

### *4.1.3 Etapa 03 - String de busca*

Foi adotada a seguinte string de busca com apenas a restrição de que os termos buscados deveriam ser identificados no título: ("introduct* programming" OR "novice programm*" OR "CS1" OR "CS 1" OR "learn* programm*" OR "programming") AND ("misunderstand*" OR "misconception" OR "difficult*" OR "mistake" OR "error"). No caso da pesquisa na IEEE Xplore, foi necessário adaptar essa string para: (("Document Title":"introduct* programming" OR "Document Title":"novice programm*" OR "Document Title":"CS1" OR "Document Title":"CS 1" OR "Document Title":"learn* programm*" OR "Document Title":"programming") AND ("Document Title":"misunderstand*" OR "Document Title":"misconception" OR "Document Title":"difficult*" OR "Document Title":"mistake" OR "Document Title":"error")).

### *4.1.4 Etapa 04 - Fontes de busca*

Consultamos três bases de dados: (1) Web of Science (WOS), por ser uma base de dados consolidada e mais seletiva em relação aos periódicos indexados (Singh et al., 2021), (2) ACM Digital Library e (3) IEEE Xplore, por serem bases de dados que indexam periódicos e conferências da área de ciência da computação, permitindo complementar os achados da WOS, principalmente em relação a artigos publicados em conferências. O fato de não termos incluído outras bases de dados como Scopus foi uma limitação deste estudo.

### *4.1.5 Etapa 05 - Coleta e análise de dados*

Coletamos os dados nas bases de dados em três momentos: na WOS e na ACM realizamos a primeira coleta no dia 04/06/2023 e a repetimos, para obter novos artigos, no dia 26/08/2024. Na IEEE, por sua vez, a coleta foi realizada no dia 02/09/2024. Ao acessar o site de cada base de dados, abrimos a busca avançada e, no campo apropriado, inserimos a string de busca. Os resultados obtidos foram integralmente exportados e organizados em uma única planilha eletrônica, na qual foram, então, analisados (consultar OMITIDO a planilha). A partir dessa planilha, seguimos o fluxo exibido na Figura 1. Duplicações de estudos foram removidas.

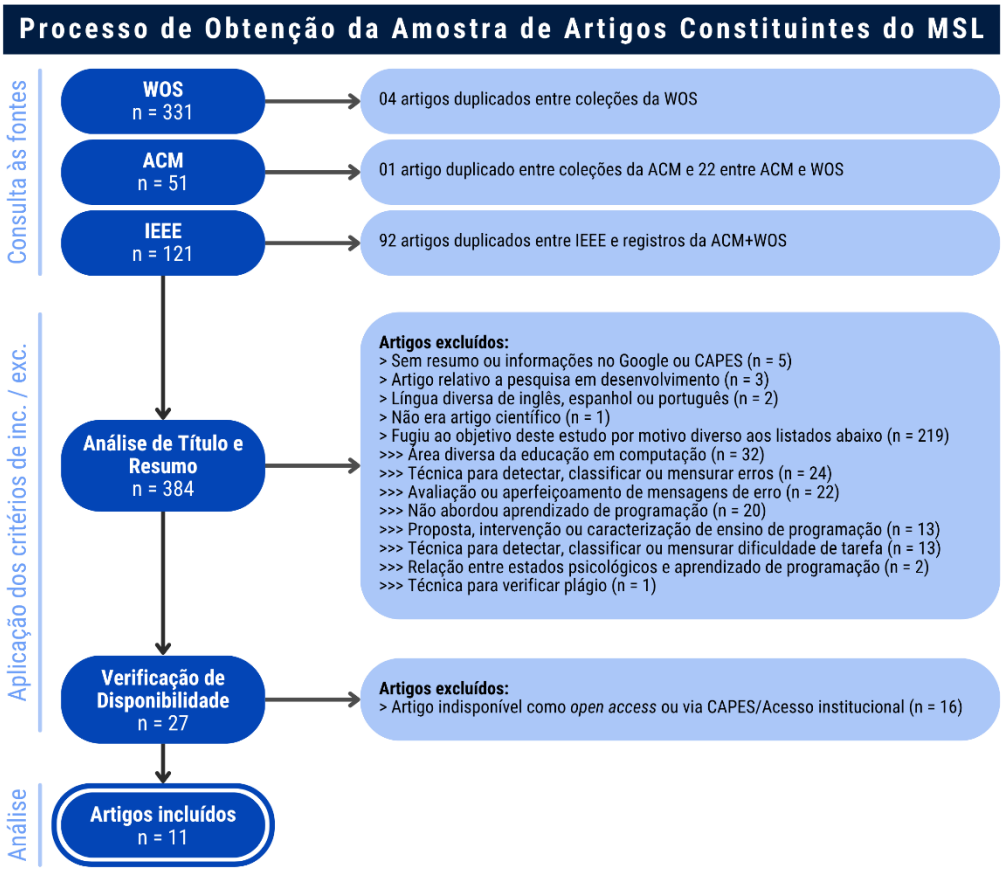


**Figura 1:** Fluxograma do processo para obtenção da amostra de artigos deste MSL.

O título de cada artigo foi lido para examinar se atendia ao critério de investigar dificuldades no aprendizado de introdução à programação. Quando havia dúvida, o resumo do artigo era consultado. Para os trabalhos aprovados nesse primeiro crivo, seguimos para a fase de *download* dos textos. Nesse processo, detectamos trabalhos que não estavam disponíveis, ou não atendiam ao critério de língua definido ou que não foram publicados em periódico, conferência ou capítulo de livro (ex.: publicação em magazine, sendo um artigo de opinião).

Fizemos o download apenas dos artigos que atenderam a todos os critérios de inclusão, os quais foram, então, lidos na íntegra. Quando a leitura evidenciava que de fato se tratava de trabalho completo e examinando dificuldades de estudantes no aprendizado de introdução à programação de computadores, registrávamos seus dados para análise e realizávamos anotações sobre as dificuldades reportadas no estudo. Essas anotações enfatizavam conceitos e conclusões relativas a quais são as dificuldades no aprendizado de programação e/ou por quais motivos aprender programação pode ser difícil. Caso a leitura indicasse que se tratava de trabalho em desenvolvimento ou que não apresentava dados explícitos sobre dificuldades no aprendizado de programação, faríamos apenas o registro do motivo identificado para a exclusão do artigo.

**Tabela 1:** Categorias de dificuldades com programação e definições correspondentes.

| Id | Nome da categoria | Definição |
|---|---|---|
| 01 | Cognição e Resolução de problemas | Características cognitivas da pessoa, incluindo raciocínio lógico, conhecimento e experiências com resolução de problemas, atitudes e estratégias diante de problemas. |
| 02 | Domínio de inglês ou outra língua | Conhecimento da língua inglesa ou outra, que seja necessária para o entendimento de termos das linguagens de programação e de mensagens de erro. |
| 03 | Conceitos de computação | Conceitos sobre como um computador funciona e que ajudam a entender como o código de um programa é executado pela máquina. |

| | | |
|---|---|---|
| 04 | Conceitos de programação | Conceitos como estruturas condicionais e laços de repetição, paradigma de programação e outros afins. |
| 05 | Conceitos relativos ao problema que precisa ser resolvido | Conhecimento de mundo necessário para resolver o problema, o que pode requerer, por exemplo, conceitos de matemática e/ou de outras áreas. |
| 06 | Domínio do Ambiente de desenvolvimento | Conhecimento dos recursos e requisitos para uso adequado do ambiente de desenvolvimento. |
| 07 | Sintaxe da linguagem | Conhecimento da estrutura que o código deve assumir para que possa funcionar, tais como a necessidade de indentação e o modo correto de escrever um comando. |
| 08 | Semântica da linguagem | Conhecimento do significado e dos usos possíveis para os comandos e recursos de uma linguagem de programação. |
| 09 | Formulação do problema | Grau de qualidade do comportamento de caracterizar o problema que precisa ser solucionado e seus requisitos. |
| 10 | Formulação de algoritmos | Grau de qualidade do comportamento de formular algoritmos. |
| 11 | Leitura e Codificação | Grau de qualidade do comportamento de codificar e ler código com compreensão. |
| 12 | Identificação de erros | Grau de qualidade do comportamento de identificar e compreender erros em código (ex.: mensagens de erro). |
| 13 | Avaliação de programa | Grau de qualidade do comportamento de avaliar adequação do programa aos requisitos. |
| 14 | Comportamento de estudo | Grau de qualidade do comportamento de estudar, envolvendo disciplina, planejamento e organização. |
| 15 | Características psicológicas | Experiências aversivas relacionadas ao contexto da programação, motivação para estudar programação e hábitos que podem impactar no aprendizado. |
| 16 | Contexto de vida e de ensino | Aspectos da vida do estudante, dentro e fora da sala de aula, que podem dificultar a sua atenção ao estudo de programação. |

As análises de dados para responder às perguntas PP01 e PP02 envolveram apenas estatísticas descritivas (contagem de frequência, cálculo de proporção, média e desvio padrão). Já para a PP03 adotamos duas abordagens de análise: considerando as anotações que fizemos ao ler os artigos sobre as dificuldades com aprendizado de programação, elaboramos um conjunto de 16 categorias de dificuldades e, em seguida, registramos, para cada artigo, quais dessas 16 categorias eram indicadas no estudo. A Tabela 1 exibe os nomes das categorias e suas definições.

Na segunda abordagem, transformamos cada anotação, de cada estudo, em um termo que consideramos representar o resultado anotado. Obtivemos um conjunto de 102 termos, sendo: 45 sobre habilidades técnicas de programação e de resolução de problemas que o estudante precisa desenvolver e que, quando ausentes, geram dificuldades no aprendizado; 10 sobre variáveis cognitivas do estudante relacionadas aos processos de raciocínio lógico, memória, atenção, representação e transferência de aprendizagem; 6 sobre variáveis psicológicas no sentido estrito de personalidade e sofrimento mental; 1 sobre variáveis de contexto da vida do estudante ou sobre sua dedicação aos estudos; 6 sobre práticas de ensino às quais o estudante é exposto; 8 sobre características da tarefa de programação; 17 sobre conceitos da computação; 3 especificamente sobre conceitos de programação orientada a objetos; e 6 sobre características da linguagem de programação. A partir desses dois procedimentos de análise, que são, na prática, representações dos achados das pesquisas examinadas, realizamos contagens de frequências.

### *4.1.6 Etapa 06 - Estratégia para garantia de qualidade*

Consideramos apenas artigos descrevendo trabalhos completos. Todos os textos foram lidos na íntegra, sendo feitas marcações no texto sobre trechos relacionados a objetivo, método e resultados e anotações em planilha acerca das dificuldades com o aprendizado de programação reportadas no estudo. Uma vez que este estudo não pretende ser uma revisão sistemática, nem realizar uma meta-análise, não estabelecemos critérios de avaliação do delineamento de cada estudo, para efeito de inclusão neste MSL. Não obstante, a nossa escolha pela Web of Science,

sem incluir outras bases que contemplam diversas áreas da ciência, foi justamente para tentar garantir a obtenção de estudos de melhor qualidade.

### 4.2 Proposição de variáveis associadas a dificuldades no aprendizado de programação e de análise do comportamento de "programar computadores com conforto e sentido"

Com base na resposta à PP03 do MSL e em nossa experiência, construímos uma proposta de lista de variáveis associadas a dificuldades, organizadas em função de se referirem a: (1) características do ambiente antecedente, (2) repertórios ausentes ou insuficientes associados às dificuldades com o aprendizado de programação e (3) características do ambiente consequente. Para essa tarefa, lemos com atenção os nossos achados e interpretamos, tendo por base conceitos analítico-comportamentais, a qual dos três tipos de categorias (antecedente, repertório ou consequente) cada achado se referia. Pela natureza interpretativa e propositiva desse exame, não conseguimos sistematizá-lo de modo que possa ser replicado. Contudo, essa proposta explicita hipóteses sobre variáveis relevantes a serem testadas em estudos futuros.

Além disso, ao ler na íntegra os artigos do MSL, ficamos atentos às referências feitas nos textos a comportamentos considerados importantes no aprendizado de programação ou que quando ausentes geravam dificuldades. Consideramos referências a termos como habilidades, competências, capacidades, aptidões e cognições como indicativos de informações sobre comportamentos. Além disso, essas referências foram identificadas nos textos porque continham verbos, referindo-se a um aprendizado importante para introdução a programação. Consideramos como válidas menções a comportamentos em todas as seções do estudo.

Para a proposição desses comportamentos (Apêndice 1), primeiro registramos todos os comportamentos-objetivo propostos por Lazzari (2013), realizando algumas adaptações textuais, para, do nosso ponto de vista, garantia de maior clareza. Em seguida, tendo por base as referências a comportamentos que encontramos ao longo da leitura dos artigos selecionados no MSL e em nossa experiência, cuidamos de adicioná-los aos comportamentos propostos por Lazzari (2013), buscando incluí-los como constituintes de uma das 7 classes mais abrangentes de comportamento propostas pela pesquisadora, conforme o seu significado. Não encontramos comportamentos que não se adequassem a uma dessas 7 classes, o que sugere que o exame de Lazzari (2013) foi suficientemente abrangente. Os comportamentos que adicionamos foram destacados por um asterisco, de modo que fique claro o que foi proposto por Lazzari (2013) e o que propusemos a partir do MSL e de nossa experiência.

Uma vez que as referências a comportamentos obtidas nos textos estavam em diversos formatos de escrita, adotamos uma adaptação do procedimento simplificado para descrição de partes funcionais de comportamentos desenvolvido por Cortegoso e Coser (2023). Esse procedimento prevê que a representação textual de um comportamento deve conter apenas 1 verbo no infinitivo, logo no início da sua descrição. Em seguida, deve ser incluído um complemento verbal que faça referência a, pelo menos, 1 evento ambiental, seja antecedente ou consequente, de modo a ampliar a clareza da descrição. Se possível, o ideal é conter os dois tipos de eventos. Contudo, para reduzir o volume textual da proposta preliminar de análise, buscamos ser tão concisos quanto possível, optando, tipicamente, pela indicação de, pelo menos, 1 evento. Este também foi um procedimento de natureza interpretativa e que não buscamos conduzi-lo de modo que pudesse ser replicado, embora seja possível realizá-lo de modo sistemático tendo essa finalidade (ver Cortegoso & Coser, 2023). Optamos por disponibilizar o produto desta análise como apêndice para permitir que no artigo pudéssemos focar nos achados do MSL e na proposição de variáveis para estudos futuros.

# 5 Resultados e Discussão

Recuperamos 503 artigos e a amostra final foi composta por 11, todos encontrados na WOS. Esse dado sugere que a WOS pode ser considerada relevante em MSLs e RSLs que investiguem dificuldades com o aprendizado de programação. Embora não tenha sido adotada nas revisões de literatura que examinamos (Qian & Lehman, 2017; Yusoff et al., 2020; Alasmari et al., 2024), pode começar a ser considerada em estudos futuros. A seguir, responderemos às três perguntas de pesquisa deste MSL.

## 5.1 Objetivo 01 - PP01. Quando e por quem os estudos são tipicamente publicados?

A Figura 2 exibe a produção de artigos ao longo dos anos e explicita uma tendência de aumento gradual nos primeiros dois períodos, seguida por um aumento grande no último, que correspondeu a 73% desta amostra. Uma hipótese para explicar esse aumento é o surgimento de muitas ações de empresas, entidades e da própria UNESCO no sentido de popularizar o ensino de programação a todas as pessoas (Unesco, 2014), bem como a disseminação do estudo e do incentivo ao ensino do pensamento computacional (Wing, 2006).

No que concerne aos pesquisadores que se destacaram com mais publicações, identificamos apenas 2, ambos com duas publicações: Qian, Y. e Lehman, J., que também publicaram juntos por duas vezes. Identificamos outros 23 pesquisadores, com uma publicação cada. Esse dado sugere que poucos grupos de pesquisadores podem estar sistematicamente envolvidos com a investigação das dificuldades com a programação. Isso pode tornar o avanço da área mais lento, sem uma comunidade para examiná-lo e organizá-lo ao longo dos anos. Não obstante, conforme discutiremos – e outros estudos já evidenciaram – os achados são robustos em torno de um mesmo conjunto de dificuldades associadas ao aprendizado de programação, mostrando que conclusões de diferentes estudos têm sido replicadas (Araújo et al., 2021).


**Figura 2:** Estudos publicados (1993 a 2024) sobre dificuldades para aprender a programar.

Assim, o consenso sobre as conclusões pode ter reduzido o incentivo para investigação do tema por diferentes grupos de pesquisadores. Argumentamos, porém, que, de um ponto de vista analítico-comportamental, apesar das replicações, ainda é preciso conhecer mais sobre o comportamento de programar computadores. Dispomos de poucos dados evidenciando quais são os comportamentos mais específicos que o constituem. Também sabemos pouco sobre em que

medida o desenvolvimento de todas as aprendizagens que já identificamos e que viermos a identificar promoverá de fato a capacidade de programar. O saneamento dessas duas lacunas no conhecimento é fundamental para mitigarmos dificuldades no aprendizado de programação e aperfeiçoarmos práticas de ensino (Kienen et al., 2021; Cortegoso & Coser, 2023).

## 5.2. Objetivo 01 - PP01. Onde os estudos são tipicamente publicados?

Com relação ao veículo de publicação dos 11 estudos examinados, obtivemos número igual de estudos publicados em conferência (de informática e/ou educação em computação ou de engenharia) e periódicos (ambos com 45%), sem destaques nos dois casos. Encontramos um estudo publicado como livro, embora no site da editora ela tenha usado o termo "*conference proceedings*". Na Tabela 2, exibimos a relação de periódicos e conferências identificadas neste estudo. Destacamos que, com base em nossos dados, os pesquisadores Qian e Lehman (2017: ACM Transactions on Computing Education (TOCE); 2022: Journal of Research on Technology in Education), únicos com mais de um artigo nesta amostra, não publicaram as duas pesquisas no mesmo local.

Observamos que nenhum periódico brasileiro ou conferência nacional apareceu em nossa investigação, nem mesmo entre os estudos identificados como adequados, mas excluídos por indisponibilidade, sendo importante avaliar em pesquisas específicas por quais motivos a produção em revistas e eventos nacionais não está indexada nem em uma base de dados que contempla diferentes áreas da ciência (WOS), tampouco em bases específicas da computação (ACM e IEEE). Também é importante que estudos futuros incluam bases de dados como a SBC Open-Lib. Esse dado corrobora achados de que a produção de artigos em outras línguas, diferentes do inglês, é pouco contemplada em bases de dados internacionais como WOS e Scopus, ocorrendo o mesmo com trabalhos publicados em conferências (Pranckutè, 2021). Nas revisões que encontramos sobre dificuldades no aprendizado de programação, não identificamos esse dado sobre veículos com maior volume de publicações. Avaliamos, contudo, que existe um equilíbrio entre publicações em periódicos e conferências. Tal resultado reflete o que é esperado da produção em ciência da computação de modo geral (Pranckutè, 2021).

**Tabela 2:** Veículos de publicação dos estudos selecionados neste MSL.

| Tipo | Veículos de publicação | Qtd. |
|---|---|---|
| Periódico | ACM Transactions on Computing Education (TOCE) | 1 |
| | Computer Science Education | 1 |
| | International Journal of Advanced Computer Science and Applications (IJACSA) | 1 |
| | International Journal of Information and Communication Technology Education (IJICTE) | 1 |
| | Journal of Research on Technology in Education | 1 |
| | **Subtotal** | **5** |
| Conferência | 2015 International Conference on Learning and Teaching in Computing and Engineering | 1 |
| | 2017 7th World Engineering Education Forum (WEEF) | 1 |
| | IEEE Frontiers in Education Conference (FIE) | 1 |
| | Proceedings of the 15th International Conference on Education Technology and Computers | 1 |
| | SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education | 1 |
| | **Subtotal** | **5** |
| Livro | Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education | 1 |
| | **Subtotal** | **1** |
| | **Total** | **11** |

## 5.3. Objetivo 01 - PP02. O que geralmente tem sido investigado?

Dos 11 estudos, 64% eram de natureza empírica, sendo que outros 2 mesclavam investigações do tipo revisão da literatura e estudo empírico. Identificamos apenas 1 estudo teórico e 1 estudo que

era exclusivamente revisão da literatura. A Tabela 3 exibe uma relação das categorias de objetivos dos estudos examinados, associada a uma breve descrição do método. Podemos notar que a maioria dos estudos (55%) buscaram caracterizar padrões de dificuldades ou erros no aprendizado de programação, envolvendo múltiplas estratégias de coleta de dados com destaque para a investigação da percepção de alunos e professores. Em seguida, com 5 estudos, destacaram-se os que caracterizaram padrões de erro, envolvendo, tipicamente, o exame de código submetido por alunos.

Chama a atenção na Tabela 3 a elevada frequência de estudos baseados apenas na percepção de estudantes e/ou professores. Esses estudos envolveram coleta de dados a partir de questionário (ocorreu em 3 estudos). O motivo do destaque é que a percepção do participante é uma fonte de informação mais sujeita a vieses, como o de desejabilidade social, bem como a erros relacionados à memória (Gouveia et al., 2009). Por exemplo, o aluno pode ter vergonha de falar sobre os próprios erros ou não lembrar adequadamente o que gerou mais dificuldades.

Além disso, parece mais relevante o fato de que o aluno pode não conhecer suficientemente o comportamento de programar computadores a ponto de identificar variáveis responsáveis pelas dificuldades que reportou, além do fato de que nem sempre temos consciência sobre o que afeta o nosso comportamento (Sério, 2005; Bandini & Delage, 2012). Por exemplo, o aluno pode identificar que não conseguia resolver um problema a partir de uma estrutura condicional, mas pode não ser capaz de indicar que aprendizagens deveria ter desenvolvido para que não enfrentasse essa dificuldade. Portanto, avaliamos que esse dado de percepção, isoladamente, além de mais frágil, pode limitar o conhecimento sobre o que de fato é crítico para as dificuldades com a programação, pois tendemos a descrever mais os "sintomas" do problema do que suas causas.

Nos estudos sobre padrões de erros e processos envolvidos no comportamento de programar, notamos a predominância de um dado mais robusto, pois está pautado em teste de desempenho do participante, ou seja, é uma tentativa de medida direta do fenômeno de interesse que é a dificuldade no aprendizado. Nesse caso, as informações foram obtidas a partir do exame de código produzido pelo participante (3 estudos), pedido de explicação de programa de computador analisado pelo participante (1 estudo) e teste objetivo de conhecimento sobre programação (1 estudo). Todavia, é preciso considerar que sem clareza de quais são todos os comportamentos que precisam ser avaliados em um teste de desempenho, corre-se o risco de identificar mais os erros típicos, como os de sintaxe, do que as dificuldades mais graves, relacionadas, por exemplo, com a resolução de problemas a partir da formulação de algoritmos, seguida pela sua tradução em código (Lazzari, 2013). Preocupa, portanto, que nenhum dos estudos analisados tenha se dedicado a um exame mais cuidadoso dos processos envolvidos no comportamento de programar computadores, o que aumenta a relevância do exame que apresentaremos neste trabalho.

**Tabela 3.** Caracterização do objetivo e método dos estudos selecionados para o MSL.

| **Categoria de Objetivo** | **Estudo** | **Síntese do método** | | | |
|---|---|---|---|---|---|
| | | **Natureza** | **Fonte sobre dificuldade** | **Participante** | |
| | | | | **Tipo** | **Curso** |
| Caracterizar padrões de dificuldades percebidas no aprendizado de programação (n = 6) | Mow (2008) | Teórico | --- | --- | --- |
| | Hashim et al. (2017) | Empírico | Percepção (questionário) | Universitário | Diversos cursos |
| | Qian e Lehman (2017) | Revisão | Literatura | --- | --- |
| | Yusoff et al. (2020) | Revisão, Empírico | Literatura, Percepção (questionário) | Professor | Computação |
| | Alasmari et al. (2024) | Revisão, Empírico | Literatura, Avaliação de software | --- | --- |
| | Rubiano et al. (2015) | Empírico | Percepção (questionário) | Professor | Computação |
| Caracterizar padrões de erros | Eranki e Moudgalya (2015) | Empírico | Teste (codificação: Java) | Universitário | Engenharia |
| | Smith e Rixner (2019) | Empírico | Teste (codificação: Python) | Diversos | --- |

| | | | | | |
|---|---|---|---|---|---|
| observados no aprendizado de programação (n = 5) | Qian e Lehman (2022) | Empírico | Teste (codificação: Python) | Aluno | Ensino Fundamental II |
| | Espinal et al. (2022) | Empírico | Teste (compreender e explicar código: MakeCode, Scratch, Python) | Aluno, Aluno | Ensino Fundamental II, Ensino Médio |
| | Yong e Tiong (2022) | Empírico | Teste (codificação: Matlab, múltipla-escolha), Percepção (questionário) | Aluno / Adolescente | Não informado |

Essa avaliação crítica não está endereçada, individualmente, aos valiosos estudos que examinamos e que, apesar de suas limitações, inerentes a qualquer pesquisa, contribuem com a investigação das dificuldades no aprendizado de programação. A crítica é no sentido do conhecimento agregado que foi produzido e tem a função de sugerir aspectos que a comunidade científica ainda precisa investigar melhor, sem prejuízo à relevância de estudos clássicos ou atuais que se dediquem a ampliar os achados sobre dificuldades percebidas ou padrões de erros, afinal também precisamos deles. Não se trata, portanto, de parar com essas pesquisas, mas de investir também na caracterização do comportamento de programar computadores, que é um tipo de estudo fundamental (De Luca et al., 2022), para que, então, possamos melhorar estratégias de medida desse comportamento e, por fim, alcançarmos dados mais robustos sobre dificuldades.

Ainda no que concerne à forma de coleta de dados, destacamos que o tipo de linguagem de programação utilizada na pesquisa e o ambiente de desenvolvimento são aspectos que podem impactar no desempenho do participante e, por sua vez, influenciar as conclusões do estudo. Com base na Tabela 3, verificamos que a linguagem mais estudada foi Python (3 estudos), seguida por Java, Matlab e Scratch (todos com 1 estudo). Entendemos que, caso a nossa amostra contivesse mais estudos antigos, poderíamos ter encontrado uma variedade maior de linguagens e um destaque para o Java. Contudo, com uma amostra na qual estudos dos últimos 10 anos predominaram, é coerente que Python se destaque, visto que é uma das linguagens mais utilizadas atualmente por desenvolvedores (Alasmari et al., 2024) e mais escolhidas para ensinar introdução à programação (Qian & Lehman, 2022).

Por fim, com relação aos participantes das pesquisas, notamos na Tabela 3 a estudos com públicos diversos, como universitários de variados cursos, tais como computação e engenharia, bem como alunos do ensino básico. De um modo geral, notamos na leitura dos artigos que a descrição dos participantes é genérica. Nem sempre temos acesso a informações como tamanho da amostra ou escolaridade dos participantes. Isso impacta negativamente na qualidade do conhecimento que está sendo construído, visto que os padrões de dificuldades ou erros e os motivos para que ocorram podem ser distintos em função de características sociodemográficas dos participantes (Hashim et al., 2017).

Vale lembrar que selecionamos neste estudo pesquisas que trabalharam com universitários, mas também com crianças/adolescentes. Seguramente, o que deve ser ensinado de programação e o repertório de entrada de cada aluno variam muito em função da idade. Nesse sentido, sem prejuízo da crítica à existência de poucos estudos sobre processos constituintes do comportamento de programar, podem ser úteis mais investigações de caracterização de dificuldades percebidas e de padrões de erros com populações diferentes da universitária ou, pelo menos, com universitários de outros cursos. É necessário que, nesses estudos, as características dos participantes sejam descritas com mais detalhes, especificando, por exemplo, quantos participantes foram avaliados, qual idade, gênero, escolaridade e, principalmente, conhecimento prévio sobre programação. Tais estudos podem combinar estratégias de coleta de dados, envolvendo avaliação de percepção e teste de desempenho, conforme feito por Yong e Tiong (2022), para que obtenham conclusões mais robustas.

Talvez, uma das dificuldades relacionadas com a descrição pormenorizada dos participantes tenha relação com o tamanho das amostras (média de 128,17 participantes por estudo empírico; *DP* = 108,49) e, principalmente, com o contexto de coleta de dados, tais como salas de aula e dados obtidos em sistemas *online* para ensino de programação. Nesses cenários, os dados disponíveis sobre os alunos podem ser reduzidos ou difíceis de coletar. Ademais, uma preocupação dos pesquisadores pode ser a de que precisam de amostras grandes para obter conclusões dos estudos com maior chance de generalização. Nessa perspectiva, a partir de contribuições da Análise do Comportamento, sugerimos que os pesquisadores considerem a possibilidade de conduzir também estudos orientados pelo delineamento de sujeito único, no qual poucos participantes sejam avaliados, tendo como controle não um grupo, mas o comportamento de cada participante (Sampaio et al., 2008). Isso significa dizer que seria examinado, em relação a cada participante, o efeito da manipulação da variável independente sobre uma dependente.

Um exemplo de estudo seria testar o efeito do ensino do comportamento de resolver problemas sobre o desempenho em um teste de codificação, no qual mais do que o código apenas, seja avaliado o processo que levou à produção do código, em que medida esse código resolveu o problema e que tipos de erros foram mais frequentes. Para cada sujeito, seria possível avaliar o desempenho antes e após o ensino. Em síntese, a ideia é que com menos pessoas sendo avaliadas na pesquisa, seria mais factível, por exemplo, coletar informações detalhadas sobre seus repertórios e como se comportam no contexto do aprendizado de programação em função de uma ou mais manipulações de condições de ensino que o pesquisador considere relevante testar.

## 5.4. Objetivo 01 - PP03. Quais são as dificuldades mais comuns no aprendizado de programação?

Com base na primeira estratégia de análise que formulamos, verificamos na Tabela 4 que, entre os 11 artigos analisados, as cinco categorias que mais produzem dificuldades em disciplinas e cursos de introdução à programação são estas: (1) sintaxe da linguagem, (2) conceitos de programação, (3) semântica da linguagem, (4) como resolver problemas (e processos cognitivos correspondentes), e (5) identificação de erros. Esses dados corroboram achados prévios de outras revisões da literatura (Qian & Lehman, 2017; Yusoff et al., 2020; Araújo et al., 2021; Alasmari et al., 2024). Avaliamos que essas cinco categorias sintetizam parte significativa das dificuldades no aprendizado de programação (Castro & Tedesco, 2020).

**Tabela 4:** Frequência de ocorrência das categorias de dificuldades com programação.

| Dificuldades no aprendizado de programação | | Qtd. | % |
|---|---|---|---|
| **Categoria** | **Estudos com ocorrência da categoria** | | |
| Sintaxe da linguagem | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Smith e Rixner (2019), Espinal et al. (2022), Yusoff et al. (2020), Qian e Lehman (2021), Yong e Tiong (2022), Alasmari et al. (2024) | 11 | 100 |
| Conceitos de programação | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Qian e Lehman (2021), Espinal et al. (2022), Yong e Tiong (2022), Yusoff et al. (2020), Alasmari et al. (2024) | 10 | 91 |
| Semântica da linguagem | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Qian e Lehman (2017), Smith e Rixner (2019), Yusoff et al. (2020), Qian e Lehman (2021), Espinal et al. (2022), Yong e Tiong (2022) | 9 | 82 |
| Cognição e Resolução de problemas | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Yusoff et al. (2020), Qian e Lehman (2021), Yong e Tiong (2022) | 8 | 73 |
| Identificação de erros | Mow (2008), Eranki e Moudgalya (2015), Hashim et al. (2017), Qian e Lehman (2017), Smith e Rixner (2019), Yusoff et al. (2020), Yong e Tiong (2022), Alasmari et al. (2024) | 8 | 73 |
| Leitura e Codificação | Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Yusoff et al. (2020), Yong e Tiong (2022), Espinal et al. (2022), Alasmari et al. (2024) | 7 | 64 |
| Características psicológicas | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Qian e Lehman (2021), Yong e Tiong (2022) | 7 | 64 |
| Formulação do problema | Mow (2008), Rubiano et al. (2015), Qian e Lehman (2017), Yusoff et al. (2020), Espinal et al. (2022) | 5 | 45 |

| | | | |
|---|---|---|---|
| Contexto de vida e de ensino | Mow (2008), Eranki e Moudgalya (2015), Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017) | 5 | 45 |
| Conceitos relativos ao problema que precisa ser resolvido | Rubiano et al. (2015), Hashim et al. (2017), Qian e Lehman (2017), Yusoff et al. (2020) | 4 | 36 |
| Formulação de algoritmos | Rubiano et al. (2015), Yusoff et al. (2020), Espinal et al. (2022), Yong e Tiong (2022) | 4 | 36 |
| Conceitos de computação | Mow (2008), Hashim et al. (2017), Yusoff et al. (2020), Espinal et al. (2022) | 4 | 36 |
| Avaliação de programa | Qian e Lehman (2017), Smith e Rixner (2019), Yusoff et al. (2020) | 3 | 27 |
| Domínio de inglês ou outra língua | Rubiano et al. (2015), Qian e Lehman (2017), Espinal et al. (2022) | 3 | 27 |
| Domínio do Ambiente de desenvolvimento | Mow (2008), Rubiano et al. (2015), Hashim et al. (2017) | 3 | 27 |
| Comportamento de estudo | Eranki e Moudgalya (2015), Hashim et al. (2017) | 2 | 18 |

Embora não tenha se destacado entre as cinco categorias mais frequentes, a categoria de "leitura e codificação" é crucial, pois abarca as categorias sobre conceitos de programação, sintaxe e semântica. Como se refere a processos comportamentais mais amplos, pode ser mais útil para pensarmos em fontes de dificuldade do que olhar, por exemplo, apenas para uma categoria como sintaxe. O motivo dessa afirmação é que, embora erros de sintaxe sejam mesmo frequentes, eles são esperados como parte do processo de aprendizado de uma linguagem, sendo mais evidente como auxiliar alunos com essa dificuldade porque a entendemos melhor. Podemos, por exemplo, usar linguagens visuais como início do aprendizado de programação ou adotar um ambiente de desenvolvimento que sinalize erros de sintaxe conforme o usuário codifica, antes mesmo de executar o seu código (Alasmari et al., 2024). Podemos, ainda, melhorar as mensagens de erro exibidas para o usuário e usar IA para fornecer ajuda adicional (Alasmari et al., 2024). Em síntese, não parece ser a sintaxe o problema que leva a reprovações e desistências em relação ao aprendizado de programação. Essa parece ser uma dificuldade adicional.

Por outro lado, parece que dispomos de menos conhecimento sobre por qual motivo alunos, por exemplo, não conseguem "enxergar uma solução", "traduzir em estruturas de programação uma ideia de solução do problema", "equivaler a ideia de que colunas e linhas de uma matriz precisam ser avaliados com uma estrutura aninhada de laços de repetição" ou "transformar problema textual em fórmula matemática que resolva determinado problema". Esses são aspectos que precisam ser mais investigados e por diferentes perspectivas teóricas, para que tenhamos mais chance de encontrar respostas que nos ajudem a aperfeiçoar o ensino de programação. Lembramos, ainda, que a categoria de "identificação de erros" também é vital, afinal os erros são a norma no desenvolvimento de programas.

**Tabela 5:** Frequência de ocorrência de termos relativos a dificuldades com programação.

| **Termo [Ocorrências]** | **01** | **02** | **03** | **04** | **05** | **06** | **07** | **08** | **09** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Linguagem: sintaxe [11] | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Habilidade: desenvolver programa para realizar uma tarefa (codificar) [09] | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Linguagem: semântica [09] | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Habilidade: depurar código [08] | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| Habilidade: projetar solução / formular algoritmo [06] | | | ✓ | | ✓ | | ✓ | | ✓ | ✓ | ✓ |
| Habilidade: resolver problemas [05] | | ✓ | ✓ | | ✓ | | ✓ | | | ✓ | |
| Habilidade: conhecer fundamentos de computação (máquina notacional) [05] | | | | ✓ | ✓ | | ✓ | ✓ | ✓ | | |
| Habilidade: corrigir erros no código [04] | | ✓ | | | ✓ | | ✓ | | | ✓ | |
| Conceito: conceitos de programação versus concepções equivocadas [04] | | | | | ✓ | | ✓ | ✓ | | | ✓ |
| Conceito: variável (declaração, uso, inicialização e tipo) [04] | | | | | ✓ | | ✓ | | ✓ | ✓ | |
| Habilidade: aplicar conceitos de programação [04] | | ✓ | | | ✓ | | ✓ | | ✓ | | |
| Habilidade: abstrair [04] | | ✓ | ✓ | | | | ✓ | | | ✓ | |

| 90 termos com menos de 4 ocorrências [129] | 04 | 07 | 06 | 04 | 11 | 03 | 12 | 05 | 07 | 09 | 05 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Termos selecionados pelos pesquisadores por sua relevância embora tenham sido menos recorrentes** | | | | | | | | | | | |
| Cognição: raciocínio lógico [03] | | | ✓ | ✓ | | | ✓ | | | | |
| Habilidade: conhecer fundamentos de matemática [03] | | | ✓ | | ✓ | | ✓ | | | | |
| Habilidade: avaliar atendimento aos requisitos do programa [03] | | | | | ✓ | ✓ | ✓ | | | | |
| Habilidade: ler código fonte com compreensão [03] | | | ✓ | | | | | | ✓ | | ✓ |
| Conceito: loop / laço de repetição [03] | | | | | ✓ | | | | ✓ | ✓ | |
| Psicológico: motivação / interesse insuficiente [03] | | ✓ | ✓ | ✓ | | | | | | | |

***Nota.*** 01 = Mow (2008), 02 = Eranki & Moudgalya (2015), 03 = Rubiano et al. (2015), 04 = Hashim et al. (2017), 05 = Qian e Lehman (2017), 06 = Smith e Rixner (2019), 07 = Yusoff et al. (2020), 08 = Qian e Lehman (2021), 09 = Espinal et al. (2022), 10 = Yong e Tiong (2022), 11 = Alasmari et al. (2024).

Por fim, as categorias com menos estudos, longe de indicarem que são aspectos menos importantes, ressaltam lacunas no conhecimento científico que precisam ser exploradas em estudos futuros. Qian e Lehman (2021), por exemplo, mostraram que adolescentes chineses, tanto de escolas típicas e de alto desempenho, cometeram mais erros de sintaxe em seus programas em razão de usarem caracteres chineses na digitação. Além disso, os alunos de escolas típicas, que tinham menos contato com computadores, apresentaram dificuldades no comportamento de digitar, uma capacidade mais básica, mas que tem potencial para gerar dificuldades no aprendizado de programação, favorecendo erros de digitação e um tempo maior para a codificação. Outro exemplo são hábitos comunicacionais do cotidiano que podem ser reproduzidos ao codificar e isso pode levar a erros, sendo difícil para o aluno evitar a emissão desses comportamentos. Nesse sentido, Qian e Lehman (2021) identificaram que hábitos relativos à resolução de problemas de álgebra também podem impactar negativamente o aprendizado de programação. Ou seja, há muito para ser investigado até que se possa criar um quadro mais consistente sobre quais aprendizagens, se desenvolvidas, podem levar à ocorrência do comportamento de programar computadores.

Na Tabela 5, apresentamos os achados em relação aos 102 termos que extraímos dos 11 estudos examinados neste MSL, destacando apenas aqueles que tiveram, no mínimo, 4 ocorrências, por economia de espaço. Trata-se de uma visão mais detalhista sobre os dados da Tabela 4. Com essa perspectiva dos nossos achados, destacam-se como as 5 principais dificuldades sintaxe da linguagem, habilidade de desenvolver programa para realizar uma tarefa, semântica, a habilidade de depurar código e a habilidade de projetar solução / formular algoritmo. Esses dados confirmam os achados apresentados na Tabela 4 e aqueles já descritos na literatura (Qian & Lehman, 2017; Yusoff et al., 2020; Araújo et al., 2021; Alasmari et al., 2024). É importante lembrar que tanto as categorias da Tabela 4, quanto os termos da Tabela 5 possuem sobreposições entre si. A habilidade de resolver problemas, por exemplo, requer raciocínio lógico e para desenvolver programa é preciso saber a sintaxe de uma linguagem de programação. Embora limitadas, Tabelas 4 e 5 sintetizam achados para explicitar o que já é bem conhecido e o que precisamos estudar mais em pesquisas futuras. Por fim, destacamos alguns termos com menor frequência, mas que se mostraram importantes na literatura.

## 5.5. Objetivo 02 - Proposta de variáveis antecedentes associadas a dificuldades no aprendizado de programação

A Figura 3 exibe variáveis relativas ao ambiente antecedente que, a partir da resposta à PP03 e da nossa experiência, avaliamos que se relacionam com dificuldades no aprendizado de programação. Para cada variável, relacionada a características ou propriedades de estímulo ou repertório, apresentamos uma definição e exemplos para ilustrá-la. Essas variáveis são úteis para que professores e pesquisadores avaliem o impacto que produzem sobre o aprendizado de programação. Vamos comentar algumas delas para demonstrar sua aplicação. Por exemplo, a variável “estímulos concorrentes” lembra que, tanto no contexto da educação básica, quanto do

ensino superior, são comuns estímulos que concorrem com a tarefa de programação. Podem ser trabalhos de outras disciplinas que estão pendentes e precisam ser finalizados logo ou, inclusive, uma longa lista de problemas de programação que precisa ser solucionada em curto tempo. Tal condição pode tornar o aluno menos atento a cada tarefa que precisa realizar, sendo que seus erros de digitação são menos em função de "falta de atenção" (como se fosse algo do estudante) e mais em razão do excesso de tarefas que precisam ser encaminhadas simultaneamente. Problemas complexos, mesmo que decompostos, podem gerar dificuldade análoga, pois são vários os aspectos a serem resolvidos. A dificuldade central aqui é a competição de estímulos discriminativos, os quais tornam provável mais de um comportamento ao mesmo tempo, sendo que a pessoa só pode lidar com uma situação por vez.

A variável de estímulos semelhantes, por sua vez, destaca que o aprendizado da discriminação de como agir corretamente em diferentes contextos se torna mais difícil quando o estudante se depara com símbolos iguais, mas com significados ou usos distintos a depender da situação, ou com símbolos ligeiramente diferentes, mas com significados / usos diferentes. O professor deve estar atento a isso e pode planejar parte do seu ensino, especificamente, para desenvolver essas discriminações, criando atividades em que, por exemplo, o símbolo "=" seja usado para atribuição e comparação. Tais atividades precisam ser sistemáticas e são necessários testes para que se possa inferir que o estudante atingiu o desempenho esperado. Não se trata, portanto, de o professor apenas falar em sala de aula "cuidado com o uso do igual nas situações de atribuição e comparação". Também é importante que o professor ensine diferenciações entre o que se aplica ao contexto da programação e da álgebra, pois alguns símbolos podem ser compartilhados, mas tendo funções distintas em cada contexto.

**VARIÁVEIS RELATIVAS AO AMBIENTE ANTECEDENTE**

*Propriedades do contexto no qual o problema é apresentado e do problema em si, que dificultam a discriminação do que precisa ser feito e reduzem as chances de sucesso do comportamento de programar computadores.*

1) **Concorrência entre estímulos**
Quantidade de estímulos interferindo em relação ao comportamento de programar computadores. *Ex.: ruído, iluminação inadequada, duas ou mais tarefas pendentes e com o mesmo prazo de entrega.*

2) **Semelhança entre estímulos**
Quantidade de estímulos fisicamente iguais ou semelhantes com mais de um significado no contexto da linguagem de programação. *Ex.: símbolo '=' compondo comando de atribuição e de comparação de valores*. Alguns estímulos antecedentes estão correlacionados, na história de uma pessoa, com disponibilidade de reforçadores, embora no contexto de programação, novo para essa pessoa, tal associação possa não ser válida, ou seja, o significado de um estímulo já conhecido, pode ser distinto, no contexto da programação, do seu uso corrente. *Ex.: símbolo '=' como sinal de atribuição e como sinal de igualdade na matemática; uso de comunicação com metonímia na situação de programação orientada a objetos (cf. Miller & Settle, 2009).*

3) **Implicitude dos estímulos**
Grau em que o estímulo é imperceptível. *Ex.: resolução do monitor ou qualidade de uma imagem podem afetar a leitura de informação necessária para resolver o problema; uso de termos incomuns com a finalidade de dificultar a interpretação do problema; mensagens de erro genéricas, sem exemplos ou indicação de onde o erro ocorreu.*

4) **Inacessibilidade dos estímulos**
Quantidade de estímulos ausentes ou inacessíveis, pouco frequentes ou incompletos, mas que são necessários para a resolução do problema. *Ex.: ausência de exemplos do que é esperado como resultado do programa; ausência de recursos necessários para a solução do problema, como informações, ambiente adequado para desenvolvimento; dados disponíveis para a solução do problema; documentação existente sobre o sistema.*

5) **Novidade dos estímulos**
Quantidade de estímulos aos quais a pessoa não foi exposta previamente ou foi pouca exposta, não tendo um significado estabelecido para a pessoa. *Ex.: quantidade de conceitos novos ou complexos de computação e/ou programação que são requeridos para a solução do problema. Yong e Tiong (2022) sugerem que conceitos de programação são difíceis de entender por que não possuem um estímulo correspondente na vida real. Isso pode ter relação com a necessidade de aprendizado de novas classes de estímulos equivalentes, pois aquelas já aprendidas não são pertinentes no contexto da programação.*

6) **Complexidade dos estímulos**

Quantidade de estímulos que devem ser considerados em conjunto para que uma resposta adequada possa ser emitida com alta probabilidade de sucesso. *Ex.: requisitos para a solução do problema; aspectos da sintaxe e semântica da linguagem de programação; paralelização do algoritmo; informações sobre funcionamento do computador.*

7) **Extensão dos estímulos**
Medidas de tamanho do(s) estímulo(s) que precisam ser considerados na tarefa de programação, podendo ser um estímulo enorme ou vários. *Ex.: total de linhas do código-fonte; número de objetos; funções; procedimentos ou instruções que precisam ser examinados.*

8) **Operações motivacionais para resolver o problema**
Grau em que a tarefa de resolução do problema de programação foi estabelecida como reforçadora ou aversiva. *Ex.: quanto é importante para a pessoa resolver o problema; quanto ela precisa resolvê-lo; variáveis como pressão familiar, problemas de saúde ou familiares; expectativas pessimistas em relação ao futuro profissional; valor aversivo ou reforçador da tarefa de resolução de problemas.*

9) **Tempo para a realização da tarefa**
Período disponível para a solução do problema de programação. *Ex.: quanto tempo o professor forneceu em sala de aula para a solução da lista de exercícios, antes de começar a correção; quanto tempo a pessoa possui, independentemente de pedido de professor, chefe ou cliente, para a solução da tarefa.*

**Figura 3:** Características do ambiente antecedente que podem gerar dificuldades no aprendizado de programação.

As variáveis de explicitude e de ausência/suficiência, por sua vez, lembram o professor sobre a importância de que os problemas apresentados tenham todas as informações necessárias, usando termos conhecidos pelos alunos, quando o objetivo for iniciar o ensino de solução de problemas. Só quando esse comportamento tiver sido instalado é que será possível ensinar comportamentos mais complexos, como solução de problemas mal definidos ou com requisitos ambíguos. No início do ensino, porém, é importante garantir que a baixa clareza das informações sobre o problema ou a ausência delas não seja mais um aspecto com o qual o aluno deve lidar, pois isso pode tornar a tarefa mais difícil (Harangus, 2019).

No geral, essas variáveis relativas ao ambiente antecedente podem levar, por exemplo, a (a) erros semânticos, como ignorar um requisito para a solução do trabalho, e, ainda, (b) erros sintáticos, tais como esquecimento de como declarar uma variável, esquecimento de colocar o ponto e vírgula, início ilegal de expressões, confusão entre operadores de atribuição e de comparação (ex.: = versus ==) e realização de operação impossível com o tipo de dado. Aspectos como concorrência entre estímulos, complexidade da estimulação discriminativa e extensão dos estímulos podem estar associados ao que é denominado de "elevada carga cognitiva" envolvida na tarefa de programação (Eranki & Moudgalya, 2015; Qian & Lehman, 2017; Espinal et al., 2022). Esses são aspectos que merecem ser melhor investigados em estudos futuros.

## 5.6. Objetivo 02 - Proposta de variáveis concernentes ao repertório comportamental associadas a dificuldades no aprendizado de programação

A Figura 4 exibe características do repertório comportamental do aprendiz que, se não tiverem sido desenvolvidas suficientemente, podem aumentar a dificuldade que ele experimenta ao aprender programação. Representamos os repertórios como "conhecimento" por questão de economia verbal, sendo importante considerar que, na realidade, cada variável remete a conjuntos complexos de comportamentos que interagem com conteúdos e informações como "conceitos de computação" e "conceitos de programação". Isso fica explícito na análise que conduzimos sobre o comportamento de programar computadores (ver Anexo 1). Para demonstrar o uso dessas variáveis por professores e pesquisadores, destacaremos alguns desses repertórios. Um deles diz respeito à relevância do domínio da língua na qual o problema é apresentado e, ainda, do inglês, típico em linguagens de programação (ex.: comandos e mensagens de erro). O professor não pode assumir, por exemplo, que todos tenham domínio da língua portuguesa, para interpretar corretamente problemas. É prudente, portanto, trabalhar com termos e estruturas mais fáceis de interpretar, para que esta variável não seja um fator adicional de dificuldade (Harangus, 2019). Com relação ao inglês, uma alternativa seria dispor de um curso de extensão ou de recursos

pedagógicos autoinstrucionais que os alunos pudessem utilizar em caso de necessidade, relacionadas ao vocabulário tipicamente requerido ao lidar com linguagens de programação.

Consideramos também a importância de conhecimento externo à computação requerido pelo problema. Na introdução à programação, é usual que seja requisitado conhecimento matemático para resolução de problemas. Com relação a esse ponto, importa lembrar que o aprendizado de matemática também tem sido desafiador para alunos, que, muitas vezes, ficam retidos e/ou evadem em disciplinas de matemática na educação básica e no ensino superior (Gris et al., 2019). Portanto, o professor de programação deve avaliar até que ponto, em que momento e de que modo ele irá envolver esse aspecto nos problemas apresentados aos seus alunos, pois, provavelmente, será um elemento adicional de dificuldade (Medeiros et al., 2020). É útil considerar também que os Conhecimentos de 03 a 08 podem levar, caso não sejam adequadamente desenvolvidos, ao surgimento de concepções equivocadas que o professor precisará trabalhar em sala.

Essas concepções se ajustam à experiência pessoal do aprendiz, sendo difíceis de serem suprimidas (Qian & Lehman, 2017; Araújo et al., 2021). Uma estratégia para superá-las é o uso de máquinas notacionais como representações do conceito de computação que precisa ser ensinado. Essa noção de representação pode ser interpretada como uma rede de estímulos equivalentes, isto é, são estímulos que, embora distintos fisicamente, compartilham uma função comportamental comum, sendo, portanto, substituíveis entre si (Sidman, 1994). Assim, a máquina notacional pode ser ensinada como equivalente, por exemplo, ao computador ou ao funcionamento do compilador de uma linguagem de programação, o que se relaciona com o tipo de pesquisa sugerida anteriormente. Tal máquina pode ter propriedades simples, mas cruciais em relação ao comportamento conceitual que o professor pretende desenvolver.

**VARIÁVEIS RELATIVAS AO REPERTÓRIO COMPORTAMENTAL DO APRENDIZ**

*O repertório do aprendiz é influenciado pelas oportunidades de capacitação às quais foi exposto. Quando essa capacitação é ausente ou insuficiente e o repertório não foi desenvolvido, o aprendiz perceberá o problema como mais difícil.*

1) **Conhecimento da língua na qual o problema é apresentado**
   Grau de domínio da língua na qual o problema e demais dados pertinentes são apresentados. *Ex.: instrução do enunciado do problema.*

2) **Conhecimento de inglês para programação**
   Grau de domínio do inglês para leitura de nomes de comandos e de mensagens da linguagem de programação e do ambiente integrado de desenvolvimento. *Ex.: mensagem de erro geradas pelo interpretador da linguagem de programação, termos da linguagem, documentação, exemplos de código.*

3) **Conhecimento externo para a solução do problema**
   Grau de domínio de outros conhecimentos, como matemáticos ou sobre uma realidade específica, e que são requeridos para a solução completa e correta do problema. *Ex.: matemática e informações sobre o mundo.*

4) **Conhecimento sobre resolução de problemas**
   Grau de domínio sobre etapas constituintes e estratégias promissoras de resolução de problemas, especialmente, no âmbito da programação. *Ex.: decompor o problema em problemas menores, identificar problemas semelhantes, depurar erros.*

5) **Conhecimento sobre computação**
   Grau de domínio sobre como um computador funciona, como programas são executados e sobre como manejar um computador (ex.: uso do mouse e teclado). *Ex.: conceitos sobre funcionamento de um computador (máquina notacional), Sistema Operacional.*

6) **Conhecimento sobre programação**
   Grau de domínio em relação a conceitos e estratégias para programar, bem como sobre boas práticas de codificação. *Ex.: conceitos básicos como variável, estrutura de controle de fluxo, laço de repetição, função etc., bem como conhecimento sobre a correspondência entre certas estruturas de programação, como laços e condicionais, e problemas nos quais são aplicáveis (que são estratégias para viabilizar a solução de um problema por meio de programação).*

7) **Conhecimento sobre linguagem de programação**
   Grau de domínio em relação ao funcionamento e uso de uma linguagem de programação específica. *Ex.: sintaxe, semântica, paradigma, mensagens de erro, tipos de exceção, comandos, documentação.*

8) **Conhecimento sobre Ambiente Integrado de Desenvolvimento (IDE)**

Grau de domínio em relação ao funcionamento e uso de um IDE específico. *Ex.: como interpretar mensagens de alerta sobre atualizações de bibliotecas e possíveis erros no código-fonte.*

9) **Competência em relação aos conhecimentos necessários para programar**
Grau de qualidade e experiência em relação ao uso dos conhecimentos supracitados. *Ex.: disciplinas e cursos realizados, experiências profissionais, projetos desenvolvidos.*

**Figura 4:** Características do repertório comportamental que podem gerar dificuldades no aprendizado de programação.

## 5.7. Objetivo 02 - Proposta de variáveis consequentes associadas a dificuldades no aprendizado de programação

A Figura 5 exibe características do ambiente consequente que podem gerar dificuldades no aprendizado de programação. Destacamos que, do ponto de vista analítico-comportamental, a evidência empírica mostra que o aprendizado de novos comportamentos requer a ocorrência de reforçamento, sendo mais efetivo quando ocorre imediatamente após a apresentação da ação (Kienen et al., 2021). Reforçamento se refere a uma relação na qual uma consequência, cuja propriedade de reforçar pode estar vinculada a uma característica da espécie ou ter sido aprendida, produzida por uma ação e que retroage sobre essa ação que a produziu tornando-a mais forte e provável de recorrer no futuro (Moreira & Medeiros, 2018). Na educação, o reforçamento precisa ser planejado para ocorrer mediante desempenho discente que seja compatível com os comportamentos-objetivo propostos (Cortegoso & Coser, 2023). Por exemplo, em uma aula expositiva, o comportamento que o professor tem oportunidade de reforçar, provavelmente, é o de "ouvir a aula atentamente". Se essa aula tiver participação, nasce para o docente, por exemplo, a oportunidade de reforçar componentes do comportamento de resolver problemas, mas só para os alunos que participarem. Se uma resposta for requerida de toda a turma, por exemplo, por meio de uma resposta por celular a um quiz, ganha-se uma chance de que uma consequência do professor possa impactar a todos. Na prática, não sabemos *a priori* se um estímulo será reforçador para um aluno. Só saberemos que o estímulo teve essa função, observando seus efeitos sobre o comportamento do aprendiz (Moreira & Medeiros, 2018).

**VARIÁVEIS RELATIVAS AO AMBIENTE CONSEQUENTE**

*O desempenho do aprendiz é influenciado pelo que ocorre após a emissão do comportamento, o que pode torná-lo mais ou menos forte e provável de recorrer no futuro. Uma vez que o comportamento de resolver um problema não consiste apenas na apresentação da solução, é importante considerar as consequências obtidas pelo aprendiz ao longo desse processo. Quando houver pouco reforçamento, o problema pode ser percebido como menos interessante e, nesse sentido estrito, mais difícil de modo que a desistência, portanto, será mais provável.*

1) **Reforçamento comportamental**
Grau em que o aluno é suficientemente reforçado em relação às suas ações consideradas corretas, em face dos comportamentos-objetivo propostos, pela apresentação de estímulos que tornem mais forte e provável a recorrência dessas ações (reforço positivo). Destaca-se aqui a importância do reforçamento positivo de comportamentos precorrentes e das tentativas de resolução de problemas e de que a distância temporal entre a apresentação da ação e o contato com o estímulo reforçador seja a menor possível (cf. Kienen et al., 2021; Cortegoso & Coser, 2023).
*Ex.: Baixa apresentação de mensagens de estímulo à continuidade da tarefa, mesmo que um erro tenha ocorrido; Pouca apresentação de mensagens destacando as partes do problema que já foram superadas; Pouco reconhecimento docente pelo trabalho realizado; Baixo volume de reforçadores positivos, relacionados à solução de problemas, gerando menor engajamento e autoconfiança; Falta ou quantidade insuficiente de atividades práticas, principalmente, em contexto de laboratório, nas quais é possível agir e ser consequenciado de modo automático e imediato; Feedback ausente, insuficiente ou inconsistente em relação ao desempenho do estudante; Práticas de avaliação adotadas pelo professor, que podem, por exemplo, exigir mais do estudante do que a capacitação fornecida; Uso da avaliação apenas com função somativa, sem considerar a sua dimensão formativa.*

2) **Controle aversivo**
Grau em que o aluno é colocado em contato, de forma consequente ou não, às suas ações no contexto acadêmico, com estímulos aversivos, seja por punição positiva, negativa ou reforçamento negativo. Quando ocorrem de forma excessiva e não planejada, podem favorecer comportamentos de fuga e esquiva de quaisquer situações relacionadas ao estudo, o que leva a dificuldades no aprendizado.

*Ex.: Apresentação de mensagens de erro recorrentes e que o aluno não consegue compreender; Brincadeiras em sala de aula sobre o insucesso do aprendiz em relação à resolução do problema; Uso das avaliações e da possibilidade de reprovação como única estratégia para favorecer comportamentos de estudo.*

**Figura 5:** Características do ambiente consequente podem gerar dificuldades no aprendizado de programação.

Apesar das dificuldades práticas para consequenciar a todos os alunos de modo suficiente e imediato, é certo que, na ausência de reforço, o aprendizado não ocorre. Nesse contexto, recursos tecnológicos podem ser muito úteis, desde jogos educativos até tarefas práticas de programação no computador ou tarefas desplugadas, mas que possam ser consistentemente acompanhadas pelo professor e/ou seus monitores e/ou por recursos informatizados, capazes de fornecer *feedback* adequado, automático e imediato (Moreira & Medeiros, 2018). Com relação ao uso de controle aversivo, embora comum nas salas de aula, é preciso avaliá-lo com atenção e cuidar para que não gere efeitos deletérios sobre saúde, aprendizado e permanência dos estudantes na instituição de ensino (ver Sidman, 2009). É usual, por exemplo, que provas sejam adotadas como recursos para garantir que o aluno estude. Essa é uma prática coercitiva socialmente aceita. São casos como esse que sugerimos que devem ser examinados com cautela em relação aos seus efeitos. No que tange a práticas preconceituosas, de desrespeito ou agressão aos alunos e, inclusive, de qualificação da programação como "intrinsecamente difícil ou para poucos", é preciso destacar que produzem repercussões negativas sobre as pessoas e não devem ser aceitas (ver Silva et al., 2023).

Complementando as Figuras 4 e 5, é importante lembrar que duas variáveis adicionais, complexas e com aspectos antecedentes e consequentes, podem impactar negativamente sobre o desempenho de alunos e professores: (1) Infraestrutura da instituição de ensino: refere-se aos recursos físicos necessários para a realização do processo de ensino-aprendizagem. Por exemplo: computadores defasados para estudar; problemas com a disponibilização de informações importantes para os aprendizes; falta de sala de aula apropriada, com conforto, segurança e condições para a devida concentração; falta de recursos para que os professores desenvolvam o seu trabalho, como tempo adequado para que possam planejar aulas, atender alunos e corrigir avaliações; (2) Organização didático-pedagógica de curso ou escola: diz respeito ao entendimento compartilhado pela equipe pedagógica sobre quais são as aprendizagens que devem ser garantidas e quais são as boas práticas para desenvolvê-las. Na ausência de organização, pode ocorrer competição de tempo de estudo ou atenção do aluno entre disciplinas muito exigentes, mas que compõem um mesmo período letivo. Pode ocorrer também variação excessiva em relação a como o ensino é realizado em cada disciplina, o que também favorece a competição entre componentes curriculares; ausência de consensos mínimos entre os docentes em relação aos comportamentos-objetivo das disciplinas pode fazer com que aprendizagens de uma disciplina, que são pré-requisitos para outra, não sejam desenvolvidas. Ainda entram nesse contexto a ausência de boas práticas de ensino, como inexistência de padrões de qualidade de planejamento, o que pode facilitar que alguns professores, por exemplo, forneçam tempo insuficiente para que os alunos aprendam um comportamento. Em síntese, as condições de ensino também podem ser fonte geradora de dificuldades para o aprendizado de programação (Silva et al., 2019).

Consideramos necessário ressaltar que, dificilmente, um professor isoladamente conseguirá produzir impacto significativo sobre grandes contingentes de alunos, pois a complexidade do processo de desenvolvimento de repertórios, tipicamente, requer múltiplas e sistemáticas experiências ao longo de várias disciplinas, para que resultados mais robustos sejam produzidos. Por isso é importante que os professores atuem em conjunto (Hattie, 2015). De todo modo, a gestão da instituição de ensino, sociedade e governo possuem responsabilidades no processo educativo, que precisam ser cumpridas para que ele atinja todo o seu potencial. Não podemos, portanto, supor como adequado que todas as responsabilidades recaiam sobre um professor ou um grupo de professores, afinal o desempenho desses profissionais depende do contexto de trabalho que é oferecido para eles (Kienen et al., 2021; Cortegoso & Coser, 2023).

# 6 Conclusão

O objetivo deste estudo foi (1) identificar dificuldades de alunos no aprendizado inicial de programação a partir de um mapeamento sistemático da literatura (MSL) e (2) propor variáveis, baseadas na teoria psicológica analítico-comportamental e passíveis de investigação por pesquisas futuras, associadas a essas dificuldades. Com relação ao "Objetivo 1 - PP01. Quando, por quem e onde os estudos são tipicamente publicados?", verificamos que a maior parte dos estudos se concentram na última década, que apenas 2 pesquisadores se destacaram tendo mais de uma publicação e que os estudos são, tipicamente, publicados em periódicos ou conferências. Com relação ao "Objetivo 1 - PP02. O que geralmente tem sido investigado e como?", notamos que os estudos, geralmente, são empíricos e caracterizam padrões de dificuldades a partir da percepção de estudantes ou professores. Por fim, sobre o "Objetivo 1 - PP03. Quais são as dificuldades mais comuns no aprendizado de programação?", encontramos dados que corroboram achados na literatura em relação a dificuldades com a linguagem e os conceitos de programação, bem como em relação a resolução de problemas, leitura e interpretação de código e identificação da fonte de erros.

No geral, os achados do MSL indicam que ainda existe muito a ser investigado sobre o aprendizado de programação e as dificuldades envolvidas nesse processo. Precisamos buscar por dados mais robustos, não só aqueles baseados em percepções. Também precisamos de dados mais específicos, não apenas sobre erros mais comuns de sintaxe, mas de dificuldades com a resolução de problemas de fato, dispondo de informações sobre repertório de entrada do estudante e suas características sociodemográficas. Em resposta ao Objetivo 2, com relação a variáveis específicas que, de um ponto de vista analítico-comportamental, podem ser investigadas, organizamos os achados da MSL em varáveis associadas aos ambientes antecedente e consequente, bem como repertórios específicos. Ressaltamos que neste estudo buscamos expor ideias coerentes com a Análise do Comportamento, mas cuja responsabilidade é nossa.

## 6.1 Implicações educacionais

Apresentamos, ao examinar as variáveis associadas às dificuldades com a programação, exemplos de como professores podem melhorar o seu planejamento de ensino. Com a análise do comportamento de "programar computadores com conforto e sentido", ilustramos a complexidade dele e explicitamos quais aprendizagens parecem ser requeridas para o seu completo desenvolvimento. Em conjunto, essas 2 fontes de contribuições podem indicar para docentes o que devem ensinar e que aspectos podem considerar sobre estímulos antecedentes e consequentes e sobre repertórios dos alunos, para atenuar dificuldades no aprendizado de programação.

Um aspecto prático que se pode levar deste estudo é que ensinar requer precisão sobre o tipo de impacto que esperamos produzir nos alunos. Quando queremos, a um só tempo, ensinar o aluno a pensar criticamente, resolver problemas, raciocinar matematicamente e computacionalmente, programar e trabalhar em equipe, aumentamos o risco de insucesso, se não conhecermos bem aspectos como o que está envolvido no ensino de cada um desses comportamentos, que dificuldades podem estar associadas a eles e para as quais devemos nos preparar, quem são nossos alunos em termos de repertório comportamental, interesses e condições para estudar e como podemos mensurar adequadamente o desempenho deles, de modo que possamos identificar o que sabiam quando o ensino começou e em que a experiência da disciplina os modificou. É, especialmente, importante examinar quais são os contextos e desafios com os nossos alunos precisam aprender a lidar para obtermos a partir disso respostas mais promissoras sobre quais comportamentos precisam aprender relativos à complexa classe geral de programar computadores com conforto e sentido.

## 6.2 Limitações do estudo e trabalhos futuros

Este estudo teve limitações que devem ser superadas em pesquisas futuras. Com relação ao Objetivo 1, a primeira diz respeito à seleção das bases de dados. Verificamos ser necessário incluir bases brasileiras, pois nem sempre revistas e eventos nacionais são indexados pelas bases internacionais. Outra limitação foi a *string* de busca. Notamos que ela retornou muitos estudos sobre técnicas para mensuração automatizada de dificuldade de uma questão ou para detecção de erros, que não eram o nosso foco. Observamos também, após a coleta, que alguns estudos identificados por revisões de literatura da nossa amostra não haviam sido recuperados por meio da nossa *string*. Assim, seria útil que estudos futuros aperfeiçoassem a nossa *string*. Considerando que nossa amostra foi pequena, é importante também que estudos futuros avaliem a possibilidade de consultar pesquisas que podem não estar disponíveis para o acesso institucional brasileiro ou como *open access*, mas que se encontram disponíveis em outras fontes como o ResearchGate. Além disso, para complementar os resultados obtidos nas bases de dados, seria importante ter olhado os artigos citados por cada estudo selecionado, pois ali podemos encontrar mais pesquisas. Apesar dessas limitações, avaliamos que nossos achados corroboram pesquisas anteriores, mostrando consistência nos dados, e contribuíram com a literatura em função da pesquisa na WOS, da inclusão de estudos com diferentes populações e do exame analítico-comportamental que foi conduzido, conferindo maior densidade à caracterização das dificuldades no aprendizado da programação.

Com relação ao Objetivo 2, a principal limitação foi a impossibilidade de replicação, dada a natureza teórica e propositiva de nossa análise. Contudo, ela indicou que, a despeito dos dados consistentes entre as diversas revisões da literatura conduzidas, ainda há muito conhecimento a ser produzido e organizado. Primeiro sobre o que precisa ser ensinado sobre programação e as dificuldades que podem surgir nesse processo, pois ambas variam entre populações. O que se ensina para crianças da educação básica não é o mesmo feito com adultos do ensino superior, e o que se ensina para alunos de engenharia civil não é o mesmo que para cientistas da computação. Embora existam dificuldades comuns, como os erros de sintaxe, podem aparecer outros tipos em cada população, relacionadas com o grau de complexidade dos comportamentos que precisam aprender.

Segundo que a efetividade de um ensino que foi planejado é muito dependente do repertório que o estudante possui no início do processo. Por melhor que seja a aula de programação, se o estudante não sabe manusear o mouse e o teclado, poderá experimentar dificuldades que, talvez, sequer tenham sido consideradas pelo professor ao preparar uma aula. Da mesma forma, alunos que já tiveram experiências anteriores com programação podem apresentar facilidade para aprender conceitos de programação e, ainda assim, dificuldades diante de novas sintaxes. Terceiro que a própria escolha de uma linguagem de programação, de paradigma e de um ambiente de desenvolvimento pode impactar em termos de dificuldade. Em resumo, essa lista de variáveis poderia seguir sendo anunciada, pois são muitos os parâmetros que precisam ser investigados. Sugerimos a realização de três estudos de base analítico-comportamental: (1) Estudo sobre obre variáveis antecedentes: Avaliar efeito de formação de classe de equivalência entre diferentes formas de representação (textual, código e fluxograma / diagrama) de estrutura lógica (condicional ou de laço de repetição) sobre o desempenho na solução de problemas de programação (consultar estudo análogo: Ribeiro et al., 2021). O pressuposto aqui é que poderemos ensinar o aluno a identificar o significa de um código a partir de estímulos correspondentes na forma de texto e diagrama, o que pode ajudar na interpretação de problemas futuros; (2) Estudo sobre variáveis consequentes: Comparar efetividade sobre o aprendizado de programação entre sistema de juiz online construído de modo a reforçar comportamentos precorrentes em relação a um sistema que reforce apenas o comportamento que produz a solução do problema. A ideia nesse caso é que precisamos reforçar toda a cadeia de comportamentos associados à solução do

problema e não apenas a resposta-solução, ou seja, precisamos promover comportamentos de manipulação do ambiente e estratégias mais promissoras de lidar com problemas no lugar de olhar, por exemplo, apenas para se o código compila ou se gerou o *output* adequado. Uma investigação sistemática nesse sentido, pode nos ajudar a identificar o que é crítico reforçar quando se ensina programação; (3) Estudo sobre variáveis relacionadas ao repertório: A partir de Lazzari (2013) e do trabalho que conduzimos no Apêndice 1, descobrir e caracterizar os comportamentos constituintes do comportamento de "programar computadores com conforto e sentido", evoluindo, em seguida, para o teste empírico desse exame a partir da organização e avaliação de eficiência de um curso orientado por esse rol de comportamentos-objetivo. Importa também conduzir essa avaliação considerando diferentes realidades sociais e perfis de estudantes.

Adicionalmente, consideramos útil examinar em estudos analítico-comportamentais sobre o aprendizado de programação, considerando diferentes perfis de estudantes e realidades sociais, quais variáveis antecedentes tendem a produzir maior dificuldade de aprendizado, quais reforçadores tendem a ser mais efetivos e quais comportamentos, quando aprendidos, produzem maior impacto positivo sobre a capacidade de uma pessoa programar computadores com conforto e sentido. Nessa perspectiva de avaliação, por exemplo, do que produz mais dificuldade em termos de variáveis antecedentes, estamos pensando em estudos que de fato manipulem propriedades do problema, como mais ou menos palavras, termos mais ou menos complexos, demanda maior ou menor de conhecimento de conceitos matemáticos etc., para avaliar se existem padrões de dificuldade, mensurados pelos tipos de comportamentos apresentados diante do problema e, ainda, pelo percentual de erros na sua solução. O mesmo raciocínio se aplica ao comparar reforçadores, como pontos, reconhecimento, aprovação etc. Precisamos de um teste para examinar se existem estímulos que tendem a ser mais efetivos, mas olhando para o seu impacto sobre a frequência de comportamentos como testar alternativa de solução do problema. Não basta, portanto, perguntar sobre a percepção ou grau de satisfação do aluno.

Esperamos que as noções de Análise do Comportamento apresentadas, junto com as variáveis propostas associadas a dificuldades e os três estudos sugeridos, possam incentivar a realização de novas investigações sobre o aprendizado de programação. Enfatizamos que temos uma preocupação aplicada, isto é, atenuar as dificuldades existentes no aprendizado de programação, mas consideramos promissor, antes de ir para a sala de aula, estudar mais sobre quais são os eventos antecedentes e consequentes associados a essas dificuldades e o que deve ser ensinado quando falamos em aprender a "programar computadores". Em última análise, avanços nessa pesquisa de base poderão evidenciar quais condições de ensino têm maior potencial de sucesso em cada contexto e para cada população.

## Declaração de conflito de interesses

Os autores declaram que não há qualquer conflito de interesses relacionado à elaboração e publicação deste estudo.

## Referências

Alasmari, O. A., Singer, J., & Ada, M. B. (2024). Do current online coding tutorial systems address novice programmer difficulties? In: *Proceedings of the 15th International Conference on Education Technology and Computers (ICETC '23) (pp. 242-248)*. Association for Computing Machinery, New York, USA. https://doi.org/10.1145/3629296.3629333 [GS Search]

Araujo, A., Zordan-Filho, D., Oliveira, E., Carvalho, L., Pereira, F., & Oliveira, D. (2021). Mapeamento e análise empírica de misconceptions comuns em avaliações de introdução à programação. In: *Anais do Simpósio Brasileiro de Educação em Computação (pp. 123-131)*. Porto Alegre: SBC. https://10.5753/educomp.2021.14478 [GS Search]

Bandini, C. S. M. B., & Delage, P. E. G. A. (2012). Pensamento e criatividade. In: M. M. C. Hübner, & M. B. Moreira, *Temas clássicos da psicologia sob a ótica da Análise do Comportamento (pp. 116-128)*. Rio de Janeiro: Guanabara Koogan.

BNCC. (2018). *Base Nacional Comum Curricular:* Educação é a Base. Recuperado de www.basenacionalcomum.mec.gov.br

Botomé, S. (2015). O conceito de comportamento operante como problema. *Revista Brasileira de Análise do Comportamento*, *9*(1), 9-46. http://dx.doi.org/10.18542/rebac.v9i1.2130 [GS Search]

Brasscom. (2021). *Demanda de talentos em TIC e estratégia ΣTCEM:* Relatório de Inteligência e Informação BR12-2021-007 – v112. Recuperado de https://bit.ly/4efZUOb

Carrara, K., & Strapasson, B. A. (2014). Em que sentido(s) é radical o Behaviorismo Radical?. *Acta Comportamentalia: Revista Latina de Análisis del Comportamiento*, *22*(1), 101-115. Recuperado a partir de https://www.revistas.unam.mx/index.php/acom/article/view/48854 [GS Search]

Carvalho-Neto, M. (2002). Análise do comportamento: Behaviorismo radical, análise experimental do comportamento e análise aplicada do comportamento. *Interação em Psicologia*, *6*(1), 13-18. http://dx.doi.org/10.5380/psi.v6i1.3188 [GS Search]

Castro, F., & Tedesco, P. (2020). Promovendo a reflexão sobre o erro em disciplinas introdutórias de programação no ensino superior. *Revista Brasileira de Informática na Educação*, *28*, 150-165. https://doi.org/10.5753/rbie.2020.28.0.150 [GS Search]

Cianca, B. C., Panosso, M. G., & Kienen, N. (2020). Programação de Condições para Desenvolvimento de Comportamentos: Caracterização da produção científica brasileira de 1998-2017. *Perspectivas em Análise do Comportamento*, *11*(2), 114-136. https://doi.org/10.18761/PAC.2020.v11.n2.01 [GS Search]

Cortegoso, A. L., & Coser, D. S. (2023). *Elaboração de programas de ensino:* Material autoinstrutivo. São Carlos: EdUFSCar. [GS Search]

De Luca, G. G., Magalhães, C. N, Rauch, S. L. B., Gusso, H. L., & Kienen, N. (2022). Problemas de pesquisa em estudos de Programação de Condições para Desenvolvimento de Comportamentos. *Acta Comportamentalia: Revista Latina de Análisis del Comportamiento*, *30*(3), 423-442. Recuperado de https://bit.ly/3Zi6PlP [GS Search]

Eranki, K. L. N., & Moudgalya, K. M. (2015). Evaluation of Programming Competency Using Student Error Patterns. In: *2015 International Conference on Learning and Teaching in Computing and Engineering (pp. 34-41)*. Taipei, Taiwan. https://doi.org/10.1109/LaTiCE.2015.16 [GS Search]

Espinal, A., Vieira, C., & Guerrero-Bequis, V. (2022). Student ability and difficulties with transfer from a block-based programming language into other programming languages: a case study in Colombia. *Computer Science Education*, *33*(4), 567-599. https://doi.org/10.1080/08993408.2022.2079867 [GS Search]

Gouveia, V. V., Guerra, V. M., Sousa, D. M. F., Santos, W. S., & Costa, J. M. (2009). Escala de Desejabilidade Social de Marlowe-Crowne: Evidências de sua validade fatorial e consistência interna. *Avaliação Psicológica*, *8*(1), 87-98. Recuperado de http://bit.ly/39mRvqK [GS Search]

Gris, G., Palombarini, L. S., & Carmo, J. S. (2019). Uma revisão sistemática de variáveis relevantes na produção de erros em matemática. *Bolema: Boletim de Educação Matemática*, *33*(64), 649-671. https://doi.org/10.1590/1980-4415v33n64a10 [GS Search]

Harangus, K. (2019). Examining the relationships between problem-solving and reading comprehension skills. *New Trends and Issues Proceedings on Humanities and Social Sciences*, *6*(5), 66-74. https://doi.org/10.18844/prosoc.v6i5.4375 [GS Search]

Hashim, A. S., Ahmad, R., & Shahrul Amar, M. S. (2017). Difficulties in Learning Structured Programming: A Case Study in UTP. In: *2017 7th World Engineering Education Forum (WEEF) (pp. 210-215)*. Kuala Lumpur, Malaysia. https://doi.org/10.1109/WEEF.2017.8467151 [GS Search]

Hattie, J. (2015). *What works best in education:* The politics of collaborative expertise. London: Pearson. Recuperado de https://bit.ly/3zd0ZHI

Fontoura-Júnior, J. M., Ribeiro, P. V. S., Pereira, L. B. F., Barros, K. W. C., Souza, O. S., Lima, R. N., Almada, N. R., Moraes, M. S., & Henklain, M. H. O. (2023). Avaliação de eficiência do curso introdução prática à programação de computadores. *Revista de Ciência e Tecnologia*, *9*(1). https://doi.org/10.18227/2447-7028rct.v97586 [GS Search]

Heward, W. L., Critchfield, T. S., Reed, D. D., Detrich, R., & Kimball, J. W. (2022). ABA from A to Z: Behavior Science Applied to 350 Domains of Socially Significant Behavior. *Perspectives on Behavior Science*, *45*, 327-359. https://doi.org/10.1007/s40614-022-00336-z [GS Search]

Kienen, N., Panosso, M. G., Nery, A. G. S., Waku, I., and Carmo, J. S. (2021). Contextualização sobre a Programação de Condições para Desenvolvimento de Comportamentos (PCDC): Uma experiência brasileira. *Perspectivas em Análise do Comportamento*, *12*(2), 360-390. Recuperado de https://www.revistaperspectivas.org/perspectivas/article/view/818 [GS Search]

Lazzari, C. L. (2013). Características da classe de comportamentos 'programar computadores' como parte da capacitação de profissional da computação. [Dissertação de mestrado]. Universidade Federal de Santa Catarina. Recuperado de https://bit.ly/47lW5F5 [GS Search]

Leão, M., & Laurenti, C. (2009). Uma análise do modelo de explicação no behaviorismo radical: o estatuto do comportamento e a relação de dependência entre eventos. *Interação em Psicologia*, *13*(1), 165-174. http://dx.doi.org/10.5380/psi.v13i1.12462 [GS Search]

Medeiros, R. P., Falcão, T. P., & Ramalho, G. L. (2020). Ensino e aprendizagem de introdução à programação no ensino superior brasileiro: Revisão Sistemática da Literatura. In: *Anais do Workshop sobre Educação em Computação (WEI) (pp. 186-190)*. Porto Alegre: Sociedade Brasileira de Computação. https://doi.org/10.5753/wei.2020.11155 [GS Search]

Merchán-Rubiano, S. M., López-Cruz, O., & Gómez Soto, E. (2015). Teaching computer programming: Practices, difficulties and opportunities. In: *2015 IEEE Frontiers in Education Conference (FIE) (pp. 1-9)*. El Paso, TX, USA. https://doi.org/10.1109/FIE.2015.7344184 [GS Search]

Moreira, M. B., & Medeiros, C. A. (2018). *Princípios básicos de análise do comportamento*. Porto Alegre: Artmed, 320p. [GS Search]

Mow, I. C. (2008). Issues and Difficulties in Teaching Novice Computer Programming. In: Iskander, M. (eds), *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education (pp. 199-204)*. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8739-4_36 [GS Search]

Pereira, A., Carvalho, L., & Souto, E. (2019). Predição de evasão de estudantes non-majors em disciplina de introdução à programação. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, *8*(1), 178-187. https://doi.org/10.5753/cbie.wcbie.2019.178 [GS Search]

Petersen, K., Vakkalanka, S., Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1-18. [GS Search]

Pranckutė, R. (2021). Web of Science (WoS) and Scopus: The Titans of bibliographic information in today's academic world. *Publications*, *9*(12), 1-59. https://doi.org/10.3390/publications9010012 [GS Search]

Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education*, *18*(1). https://doi.org/10.1145/3077618 [GS Search]

Qian, Y., & Lehman, J. (2021). Using an automated assessment tool to explore difficulties of middle school students in introductory programming. *Journal of Research on Technology in Education*, *54*(3), 375-391. https://doi.org/10.1080/15391523.2020.1865220 [GS Search]

Ribeiro, K. L., Oliveira, Y. N., & Henklain, M. H. O. (2021). Treinar a correspondência entre diferentes formas de apresentar problemas melhora o desempenho matemático. *Avances en Psicología Latinoamericana*, *39*(1), 1-18. https://doi.org/10.12804/revistas.urosario.edu.co/apl/a.8931 [GS Search]

Sampaio, A., de Azevedo, F., Cardoso, L., de Lima, C., Pereira, M., & Andery, M. (2008). Uma introdução aos delineamentos experimentais de sujeito único. *Interação em Psicologia*, *12*(1), 151-164. http://dx.doi.org/10.5380/psi.v12i1.9537 [GS Search]

Santana, B. L., Chavez, C. V. F. G., & Bittencourt, R. A. (2021). Uma definição operacional para pensamento computacional. In: *Anais do Simpósio Brasileiro de Educação em Computação (EDUCOMP) (pp. 93-103)*. Porto Alegre: SBC. https://doi.org/10.5753/educomp.2021.14475 [GS Search]

Sério, T. M. de A. P. (2005). O behaviorismo radical e a psicologia como ciência. *Revista Brasileira de Terapia Comportamental e Cognitiva*, *7*(2), 247-261. https://doi.org/10.31505/rbtcc.v7i2.554 [GS Search]

Sidman, M. (1994). *Equivalence relations and behavior:* A research story. Boston: Authors Cooperative. [GS Search]

Sidman, M. (2009). *Coerção e suas implicações* (M. A. Andery & T. M. Sério, trads). Campinas, SP: Livro Pleno. (Trabalho original publicado em 1989).

Silva, D. N., Brito, J. R., & Vaz, N. A. P. (2019). Lógica de Programação: Dificuldades de ensino-aprendizagem, métodos e ferramentas computacionais. In: *Anais do X Simpósio de Tecnologia da Informação, XI Semana de Iniciação Científica do curso de Sistemas de Informação e IV Colóquio de Estágio (sem página)*. Goiás: Universidade Estadual de Goiás. Recuperado de https://www.anais.ueg.br/index.php/sti_sic/article/view/13982 [GS Search]

Silva, E., Caceffo, R., & Azevedo, R. (2022). Análise dos tópicos mais abordados em disciplinas de introdução à programação em universidades federais brasileiras. In: *Anais do II Simpósio Brasileiro de Educação em Computação (pp. 29-39)*. Porto Alegre: SBC. https://doi.org/10.5753/educomp.2022.19196 [GS Search]

Silva, G. H. G. da, Lautert, S. L., Carmo, J. S., Santos, E. M., & Santos, D. E. L. (2023). Microagressões no contexto de ensino e aprendizagem da matemática: Uma análise teórico-conceitual. *Educação Matemática Pesquisa Revista do Programa de Estudos Pós-Graduados em Educação Matemática*, *25*(1), 283-304. https://doi.org/10.23925/1983-3156.2023v25ip283-304 [GS Search]

Singh, V. K., Singh, P., Karmakar, M., Leta, J., & Mayr, P. (2021). The journal coverage of Web of Science, Scopus and Dimensions: A comparative analysis. *Scientometrics*, *126*(6), 5113-5142. https://doi.org/10.1007/s11192-021-03948-5 [GS Search]

Smith, R., & Rixner, S. (2019). The error landscape: Characterizing the mistakes of novice programmers. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19) (pp. 538-544)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3287324.3287394 [GS Search]

Todorov, J. C. (2007). A Psicologia como o estudo de interações. *Psicologia: Teoria e Pesquisa*, *23*(spe), 57-61. https://doi.org/10.1590/S0102-37722007000500011 [GS Search]

UNESCO (2014, 10 de julho). *Learn by coding*. Recuperado de https://www.unesco.org/en/articles/learn-coding

Wazlawick, R. S. (2021). *Metodologia de pesquisa para ciência da computação* (3 ed.). Rio de Janeiro: LTC. [GS Search]

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215 [GS Search]

Yong, S. T., & Tiong, K. M. (2022). A Blended Learning Approach: Motivation and Difficulties in Learning Programming. *International Journal of Information and Communication Technology Education (IJICTE)*, *18*(1), 1-16. http://doi.org/10.4018/IJICTE.301276 [GS Search]

Yusoff, K. M., Ashaari, N. S., Wook, T. S. M. T., & Ali, N. M. (2020). Analysis on the requirements of computational thinking skills to overcome the difficulties in learning programming. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *11*(3), 244-253. http://dx.doi.org/10.14569/IJACSA.2020.0110329 [GS Search]

Zilio, D., & Neves Filho, H. (2018). O que (não) há de "complexo" no comportamento? Behaviorismo radical, self, insight e linguagem. *Psicologia USP*, *29*(3), 374-384. https://doi.org/10.1590/0103-656420170027 [GS Search]

**Apêndice 01.** Proposta preliminar de análise das classes mais específicas de comportamento constituintes da classe geral "programar computadores com conforto e sentido", composta por comportamentos identificados por Lazzari (2013) e comportamentos descobertos no MSL (indicados por um asterisco).

| CLASSE GERAL DE COMPORTAMENTO |
|---|
| **1. Programar computadores com conforto e sentido.** |
| **CLASSES DE COMPORTAMENTO CONSTITUINTES DA CLASSE GERAL** |
| **1.1. Avaliar argumentos de acordo com regras lógicas.**<br>**1.1.1.** Identificar princípios básicos da lógica (ex.: princípios da identidade, não-contradição, terceiro excluído).<br><br>**1.1.2.** Elaborar argumentos de acordo com regras lógicas.<br><br>**1.2. Caracterizar funcionamento de computadores.**<br>**1.2.1.** Identificar conceitos básicos do computador para que fique explicitada a relação de dependência entre *hardware* e *software*.<br><br>**1.2.2.** Caracterizar estrutura e funcionamento dos computadores com base em seus principais componentes: processador, memória principal, memória secundária e barramento.<br>**1.2.2.1.** *Descrever o funcionamento do Sistema Operacional no sentido de como ele gerencia os recursos de *hardware* do computador e como isso impacta a execução do *software*.<br>**1.2.2.1.1.** *Descrever como um programa é executado pelo computador (máquina notacional), considerando que a máquina não pode inferir intenções do programador, devendo a comunicação com ela ser lógica, precisa e consistente a todo momento.<br>**1.2.2.1.2.** *Caracterizar como funciona a tradução do código de linguagem de programação para linguagem de máquina a partir de um compilador e de um interpretador.<br><br>**1.3. Resolver problemas.**<br>**1.3.1.** *Caracterizar no que consiste o processo comportamental de resolver problemas e como isso se aplica à programação de computadores.<br>**1.3.1.1.** *Descrever etapas e estratégias promissoras de resolução de problemas com base no conhecimento existente sobre o que deve ser considerado e testado em processos de solução de problemas de programação.<br>**1.3.1.1.1.** *Identificar que as informações sobre o problema podem ser divididas entre importantes e aquelas que podem ser, convenientemente, ignoradas para reduzir a complexidade do problema.<br>**1.3.1.2.** *Reconhecer a etapa em que uma pessoa se encontra no processo de solução de problemas como forma de, sempre que necessário, facilitar a verificação do próprio progresso na solução de um problema específico.<br>**1.3.1.3.** *Identificar dificuldades inerentes à resolução de problemas no âmbito da computação de modo a atenuar o efeito de erros e falhas no processo de programação.<br>**1.3.1.4.** *Identificar que a solução de problemas de programação requer adesão a regras lógicas estritas e comunicação inequívoca de modo a atenuar a probabilidade de erros por desconsideração desses requisitos.<br><br>**1.3.2.** Caracterizar problemas para facilitar a sua resolução.<br>**1.3.2.1.** Diferenciar demandas recebidas de problemas a resolver, para aumentar as chances de que problemas sejam derivados da demanda e, assim, o que importa e precisa ser feito seja priorizado.<br>**1.3.2.2.** *Interpretar o problema a partir das informações disponíveis.<br>**1.3.2.2.1.** *Avaliar grau de explicitude do problema e o que ainda precisa ser conhecido para compreendê-lo.<br>**1.3.2.2.2.** Identificar objetivo do problema.<br>**1.3.2.2.3.** *Avaliar em que medida o objetivo do problema que foi identificado pode estar errado, para aumentar as chances de correção de equívocos de interpretação.<br><br>**1.3.3.** Delimitar escopo do problema, considerando os seus requisitos.<br>**1.3.3.1.** *Identificar informações que precisam ser conhecidas para a solução do problema (domínio externo ao problema).<br>**1.3.3.2.** *Identificar requisitos do problema e diferenças que ele possui em relação a outros problemas já solucionados, de modo a evitar confusões entre como este problema pode ser resolvido e soluções implementadas no passado.<br><br>**1.3.4.** Representar problemas de diferentes formas, para facilitar a sua análise e solução (ex.: por meio de fluxograma, linguagem natural, linguagem matemática, representação gráfica).<br>**1.3.4.1.** Caracterizar dados de um problema.<br>**1.3.4.2.** Nomear dados de um problema.<br>**1.3.4.3.** Selecionar dados necessários para resolver um problema.<br>**1.3.4.4.** *Identificar padrões em um problema que facilitem a descoberta da sua solução.<br><br>**1.3.5.** *Avaliar o que há em comum entre problemas conhecidos e suas soluções em relação ao problema que precisa ser solucionado, para viabilizar identificação e uso de recursos, boas práticas e soluções que sejam pertinentes.<br>**1.3.5.1.** *Pesquisar problemas análogos e suas respectivas soluções.<br>**1.3.5.2.** *Pesquisar algoritmos pertinentes para o problema que precisa ser resolvido.<br><br>**1.3.6.** *Abstrair aspectos irrelevantes do problema de modo a reduzir o efeito confundidor que variáveis irrelevantes poderiam exercer, caso não tivessem sido desconsideradas.<br><br>**1.3.7.** Decompor problema em problemas de menor complexidade, com partes pequenas, únicas e não ambíguas, que são resolvidas de forma independente.<br>**1.3.7.1.** Definir critérios para decompor um problema.<br>**1.3.7.2.** Identificar partes de um problema que são, elas próprias, problemas mais específicos a serem solucionados.<br><br>**1.3.8.** *Formular hipóteses sobre solução potencial para um problema. |

**1.3.8.1.** *Identificar estratégias para a produção de soluções criativas.
**1.3.8.2.** *Derivar soluções de problemas conhecidos para o problema que precisa ser resolvido, considerando as diferenças entre ambos, os requisitos de solução do problema, as estratégias de solução válidas e a necessidade de cuidado para reduzir as chances de aplicação mecânica de solução criada para outro problema em relação ao problema atual.
**1.3.8.2.1.** *Avaliar se a combinação de duas ou mais soluções para problemas conhecidos podem ajudar a superar o problema que precisa ser resolvido.
**1.3.8.2.2.** *Inferir a partir de caso particular (ex.: casos de teste) a solução do problema.

**1.3.9.** *Testar hipóteses de solução que foram formuladas ("simulação mental").
**1.3.9.1.** *Identificar resultados dos testes de hipóteses de solução.
**1.3.9.2.** *Avaliar criticamente os resultados das hipóteses de solução testadas.
**1.3.9.2.1.** *Explicar para si mesmo a hipótese de solução como estratégia metacognitiva de exame de sua pertinência.
**1.3.9.3.** *Aperfeiçoar hipóteses com base nos resultados do teste de modo a viabilizar a melhor solução.

**1.3.10.** Escrever solução de problemas de diferentes graus de complexidade.
**1.3.10.1.** Diferenciar o que sabe do que é preciso saber fazer para resolver problemas.
**1.3.10.2.** Identificar possibilidades de solução mais promissoras para um problema, dentre as hipóteses testadas.

**1.4. Construir algoritmos.**
**1.4.1.** Diferenciar algoritmo de programa de computador.
**1.4.1.1.** Caracterizar algoritmo.
**1.4.1.2.** Caracterizar programa de computador (ex.: paradigma de programação adotado, finalidade).

**1.4.2.** Caracterizar estruturas de controle de fluxo (ex.: seleção, repetição), para aumentar as chances de que a estrutura adequada para uma situação seja identificada e que algoritmos concisos e eficientes sejam desenvolvidos.

**1.4.3.** *Caracterizar conhecimentos matemáticos (ex.: aritmética, álgebra) que são tipicamente relevantes e requeridos para o desenvolvimento de algoritmos.

**1.4.4.** Ler instruções de um algoritmo, para aumentar as chances de escrevê-los e interpretá-los corretamente.

**1.4.5.** Escrever algoritmos, considerando que devem ser compreensíveis para outros programadores.
**1.4.5.1.** Caracterizar instruções de algoritmos (ex.: declaração referencial, declaração conotativa).
**1.4.5.2.** Identificar o que cada instrução faz.
**1.4.5.3.** Identificar o que as instruções fazem juntas.
**1.4.5.4.** Definir sequência de instruções pequenas, únicas, não ambíguas e finitas que solucionem um problema.
**1.4.5.5.** *Criar estrutura de repetição quando a execução reiterada de uma sequência de instruções for necessária para a solução do problema.
**1.4.5.6.** *Criar estrutura condicional quando a mudança no fluxo de execução, baseada em regras, for necessária para a solução do problema.

**1.5. Formalizar algoritmos.**
**1.5.1.** Identificar equivalência de instruções em linguagem natural e de programação.
**1.5.1.1.** Caracterizar linguagens de programação (ex.: grau de abstração, paradigma de programação, geração).
**1.5.1.1.1.** *Identificar finalidade dos paradigmas de programação no sentido de estratégias para interpretação de problemas e suas soluções de modo a permitir a construção de programas de computador.
**1.5.1.1.2.** *Descrever os principais tipos de paradigmas de programação (ex.: estruturado, orientado a objetos).
**1.5.1.1.2.1.** *Caracterizar o paradigma orientado a objetos em termos de significado, interação e implementação de seus principais conceitos (ex.: objeto, classe, instância de classe, método, herança, polimorfismo, variável estática e interface.
**1.5.1.1.2.1.1.** *Implementar componentes do paradigma orientado a objetos (objeto, classe, instância de classe, método, herança, polimorfismo, variável estática, interface).
**1.5.1.1.2.1.2.** *Identificar origem e destino dos valores de parâmetros processados pelo método de um objeto.
**1.5.1.1.2.2.** *Caracterizar o paradigma estruturado.
**1.5.1.1.2.2.1.** *Implementar componentes do paradigma estruturado.
**1.5.1.2.** Caracterizar a sintaxe de uma linguagem de programação, considerando que se trata das regras gramaticais da linguagem.
**1.5.1.2.1.** *Identificar na sintaxe de uma linguagem a função específica para estímulos como indentação, uso de parênteses, colchetes, chaves, aspas duplas e simples, ponto e vírgula, vírgula e operadores de atribuição, comparação, aritméticos e lógicos.
**1.5.1.3.** Caracterizar a semântica de uma linguagem de programação, considerando que se trata do vocabulário da linguagem e seu significado.
**1.5.1.4.** Identificar diferentes usos de um símbolo em uma linguagem de programação (ex.: +, ++, =, ==).
**1.5.1.5.** Diferenciar uso de um símbolo em diferentes linguagens de programação.
**1.5.1.6.** *Traduzir termos técnicos da programação, no contexto de uma linguagem de programação, para a língua nativa do desenvolvedor de modo a garantir que foram adequadamente compreendidos.

**1.5.2.** Caracterizar descrições formais e precisas de problemas.
**1.5.2.1.** Identificar estruturas que favorecem a formalização de um algoritmo.
**1.5.2.2.** Identificar níveis de formalização, para aumentar as chances de escrita de códigos legíveis, organizados e corretos.
**1.5.2.3.** Identificar regras de formalização.

**1.5.3.** Especificar os detalhes formais de um algoritmo.
**1.5.3.1.** Identificar o objetivo de formalizar um algoritmo.
**1.5.3.2.** Definir o nível de formalização, considerando a linguagem de programação escolhida.

**1.6. Escrever programas de computador.**
**1.6.1.** Nomear variáveis respeitando as regras de uniformidade.
**1.6.1.1.** *Conceituar variável com base em seu sentido típico e específico, no contexto de uma linguagem de programação.
**1.6.1.1.1.** *Identificar que uma variável só pode ter um valor por vez, em um momento específico.
**1.6.1.1.2.** *Identificar que a ordem de atribuição de valores a uma variável importa (ex.: 5=A não é equivalente a A=5).

**1.6.1.1.3.** *Identificar que funções de impressão de valores na tela não podem atribuir valores a uma variável.
**1.6.1.1.4.** Diferenciar o nome atribuído a uma variável em relação ao que ela faz (ex.: o nome da variável pode ser média e a variável receber o valor da somatória de dois números).
**1.6.1.2.** Escolher nomes de variáveis coerentes com a sua função no código.
**1.6.1.2.1.** *Avaliar se o nome de uma variável é adequado em relação às boas práticas de programação.
**1.6.1.2.2.** *Avaliar se nome da variável não viola regras sintáticas da linguagem de programação utilizada.
**1.6.1.2.3.** *Avaliar se nome de uma variável não sobrescreve funções embutidas (built-in) ou palavras reservadas da linguagem utilizada.

**1.6.2.** Declarar variáveis.
**1.6.2.1.** Caracterizar variáveis em termos do seu escopo (local ou global), tipo, necessidade de inicialização e sintaxe correta para a sua declaração.
**1.6.2.1.1.** *Identificar as implicações de uma variável ser local ou global.
**1.6.2.2.** Identificar diferentes funções das variáveis (ex.: contador de repetição, seleção, soma, inicialização, atribuição, acumulação).
**1.6.2.3.** Identificar tipos de variáveis (ex.: inteiro, real, *string*).
**1.6.2.4.** Identificar o que acontece no computador quando uma variável é declarada.
**1.6.2.5.** Definir o tipo e o nome da variável.

**1.6.3.** Atribuir valores às variáveis compatíveis com o tipo da variável declarada.
**1.6.3.1.** Caracterizar atribuição de valores às variáveis.
**1.6.3.1.1.** Identificar o que acontece no computador quando valores são atribuídos a uma variável.

**1.6.4.** Caracterizar etapas do processo de programação de computadores.
**1.6.4.1.** Caracterizar termos básicos de programação e como implementá-los (ex.: regras de precedência, operadores de atribuição, comparação, aritméticos e lógicos, tipos abstratos de dados, estrutura de decisão / seleção, estrutura de loop / laço de repetição, função, procedimento, parâmetro, vetor, matriz, recursão, estruturas de dados básicas, ponteiro, arquivos).
**1.6.4.1.1.** *Identificar que, em uma estrutura condicional, ou o bloco de *if* ou o de *else* será executado e não ambos.
**1.6.4.1.2.** *Identificar o escopo do laço de repetição.
**1.6.4.1.3.** *Identificar que a execução de um programa é sequencial.
**1.6.4.2.** *Identificar necessidade de uso de bibliotecas para facilitar o desenvolvimento do programa.
**1.6.4.3.** Identificar a melhor maneira de leitura de um programa de computador.
**1.6.4.4.** *Caracterizar boas práticas de programação relacionadas ao uso de estruturas de seleção e repetição e à modularização do programa.
**1.6.4.5.** *Caracterizar estilos de programação para identificação dos mais promissores.
**1.6.4.6.** *Identificar todos os passos de execução de um programa de computador.

**1.6.5.** Projetar programas de computador.
**1.6.5.1.** Definir paradigma de programação.
**1.6.5.2.** Definir especificações do programa (ex.: o que o programa precisa resolver).
**1.6.5.2.1.** *Identificar os requisitos (ex.: funcionais e não-funcionais) que precisam ser atendidos para que o programa seja considerado completo e correto.
**1.6.5.2.2.** *Diferenciar dados de entrada dos dados de saída.
**1.6.5.3.** *Planejar o que precisa ser codificado para evitar retrabalho.
**1.6.5.3.1.** *Definir o que precisa ser codificado primeiro e o que pode ser feito na sequência.
**1.6.5.3.2.** *Criar rascunho do programa de modo a facilitar a visualização de sua estrutura e o que precisa ser desenvolvido, considerando, ainda, o algoritmo formalizado.
**1.6.5.3.3.** *Avaliar se código existente pode ser reutilizado, antes ou após ajuste, ou integrado ao programa que precisa ser desenvolvido.
**1.6.5.3.4.** *Avaliar se ferramentas (ex.: inteligência artificial/LLMS) podem ajudar na produção de código.
**1.6.5.3.4.1.** *Examinar se o uso de uma ferramenta respeita direitos autorais, legislação específica do país em que o programa está sendo desenvolvido ou regras definidas pelo cliente e aspectos técnicos como os de segurança da informação.
**1.6.5.3.5.** *Propor estratégias para tratamento de casos atípicos relacionados ao uso do programa.
**1.6.5.3.5.1.** *Prever possíveis problemas no uso do programa como pré-requisito para que seja possível o adequado tratamento desses erros (ex.: evitar divisão por zero).

**1.6.6.** *Caracterizar recursos e restrições do Ambiente Integrado de Desenvolvimento (IDE) para garantir que o programa possa ser desenvolvido com agilidade e segurança.

**1.6.7.** *Ler programas de computador com conforto e sentido.
**1.6.7.1.** *Localizar elementos de interesse dentro do código.
**1.6.7.2.** *Identificar a finalidade do programa a partir da leitura do seu código-fonte.
**1.6.7.3.** *Identificar que estruturas estáticas podem representar processos dinâmicos no código-fonte.

**1.6.8.** Escrever códigos em uma linguagem de programação.
**1.6.8.1.** Selecionar problemas possíveis de resolver com um programa de computador.
**1.6.8.2.** Identificar limitações para escrever um programa de computador.
**1.6.8.3.** *Traduzir algoritmo formalizado em termos de uma linguagem de programação.
**1.6.8.4.** *Modularizar código de modo que ele seja organizado em procedimentos ou funções.
**1.6.8.5.** *Unir blocos de código que deveriam funcionar de modo coordenado.
**1.6.8.6.** *Identificar quais linhas são executadas e quais não são dado um conjunto de entradas.
**1.6.8.7.** *Atualizar o estado das variáveis a cada linha de instrução
**1.6.8.8.** *Implementar medidas de segurança (ex.: evitar divisão por zero).

**1.7. Avaliar programas de computador.**
**1.7.1.** *Projetar casos de teste conforme o contexto do problema, para que o programa possa ser testado.
**1.7.2.** *Implementar casos de teste para exame do funcionamento do programa de computador.

**1.7.3.** Detectar possíveis erros para que sejam corrigidos.
**1.7.3.1.** *Interpretar mensagens de erro apresentadas pela linguagem de programação.

**1.7.3.1.1.** Classificar tipo de erro encontrado em um programa de computador (ex.: sintático, semântico/lógico).
**1.7.3.1.2.** *Corresponder termos na mensagem de erro a partes do código.
**1.7.3.1.2.1.** *Localizar erros no código-fonte do programa, considerando os próprios conhecimentos ou com auxílio de fóruns e de modelos de linguagem de larga escala, desde que observados aspectos éticos, jurídicos e técnicos em relação ao uso dessa tecnologia.
**1.7.3.1.3.** *Investigar com prioridade erros mais prováveis como os de sintaxe ou cometidos em função da pequena experiência com a programação (ex.: relacionados a hábitos de comunicação, hábitos de solução de problemas matemáticos).
**1.7.3.2.** Identificar efeitos de erros no programa.
**1.7.3.3.** Testar o programa elaborado para detecção de erros.

**1.7.4.** Depurar erros por meio de rastreamento e teste do programa de computador.
**1.7.4.1.** Caracterizar depuração de erros como processo de análise do código-fonte.
**1.7.4.2.** Examinar os processos que levaram ao erro por meio de rastreamento do código-fonte.
**1.7.4.2.1.** Identificar o que foi escrito ou feito de errado a partir de mensagem de erro.
**1.7.4.2.2.** *Identificar fonte de erro relacionada a um problema de compilação ou interpretação.
**1.7.4.2.3.** *Identificar fonte de erro relacionada a problema na execução do código-fonte.
**1.7.4.2.3.1.** *Conduzir, sempre que for pertinente, teste de mesa.
**1.7.4.2.4.** *Identificar fonte de erro relacionada a problema em relação aos requisitos de solução do problema.
**1.7.4.2.5.** Diferenciar o local onde o erro aparece do local em que ele se origina.
**1.7.4.3.** Corrigir erros encontrados por meio de rastreamento do código-fonte ou do uso de outras técnicas.
**1.7.4.3.1.** *Identificar ações adequadas para lidar com os tipos de erros encontrados.
**1.7.4.3.2.** *Implementar soluções para os erros identificados.

**1.7.5.** Avaliar qualidade de programas de computador.
**1.7.5.1.** Diferenciar um programa que funciona daquele que está terminado e validado.
**1.7.5.2.** Definir critérios para avaliar programas de computador (ex.: eficiência, custo, atendimento aos requisitos, documentação e legibilidade).
**1.7.5.3.** *Refatorar código-fonte sempre que possível, podendo empregar os próprios conhecimentos ou o auxílio de inteligência artificial, desde que observados aspectos éticos, jurídicos e técnicos em relação ao uso dessa tecnologia.

**Grau 01 – Classe Geral:** 01 (“Programar computadores com sentido e conforto”)
**Total de comportamentos intermediários (constituintes da classe geral):** 172
**Comportamentos intermediários propostos por Lazzari (2013):** 82
**Comportamentos intermediários descobertos no MSL (indicados por asterisco):** 90

| Total de comportamentos intermediários identificados por Lazzari (2013) e neste estudo a partir de MSL | | | | | |
|---|---|---|---|---|---|
| **Grau 02** | **Grau 03** | **Grau 04** | **Grau 05** | **Grau 06** | **Grau 07** |
| 07 | 32 | 73 | 43 | 12 | 05 |