Linguagens visuais para o ensino de programação: uma revisão da literatura com foco em paradigmas de programação

Marina Silva da Silva, Ana Paula Lüdtke Ferreira
{marinadsds2.aluno,anaferreira}@unipampa.edu.br
Universidade Federal do Pampa, Campus Bagé, Bagé/RS

RESUMO

As primeiras experiências com programação determinam a atitude desenvolvida pelos aprendizes com relação ao assunto. Frustrações e dificuldades levam à desilusão com a área de Computação para estudantes do Ensino Básico e à retenção de alunos nos primeiros semestres do ensino superior. Uma das dificuldades enfrentadas é o foco em aspectos operacionais da solução de problemas, gerando excessivo esforço na correção de erros de sintaxe e menor atenção no problema a resolver. Linguagens visuais são atrativas devido ao seu aspecto lúdico e aos ambientes de desenvolvimento que facilitam a composição de operadores, estruturas e código. Este trabalho investiga, por meio do método de revisão sistemática da literatura, as linguagens visuais usadas para ensino de programação, com foco no paradigma de programação subjacente, que determina a forma de pensar e implementar a solução de um problema. A revisão mostrou a dominância do paradigma imperativo, decisão de projeto de linguagem que tem impacto tanto na forma como os programas são pensados e construídos quanto na atitude desenvolvida pelos estudantes a respeito da programação.

CCS CONCEPTS

• **Social and professional topics** → Computing education.

PALAVRAS-CHAVE

Educação em Computação, Paradigmas de programação, Linguagens de programação

1 INTRODUÇÃO

O número de estudantes em todos os cursos da área de Computação corresponde a cerca de 5,5% dos ingressos e 4% do total de matrículas no ensino superior brasileiro [15, 40]. Contudo, a disseminação do uso de dispositivos computacionais na sociedade praticamente não mudou o contato dos estudantes da Educação Básica com atividades curriculares de programação, em sua maior parte restrito a poucas escolas particulares.

O primeiro semestre dos cursos de graduação apresenta os maiores índices de retenção em relação aos semestres seguintes, fato frequentemente atribuído a uma suposta "falta de base" em relação aos conhecimentos provenientes do Ensino Básico. Essa justificativa pode parecer verdadeira, mas não é suportada por dados de desempenho discente. Primeiramente, alunos com notas altas no ENEM apresentam as mesmas dificuldades com relação à programação dos alunos que ingressam com notas inferiores. Esse fenômeno pode ser explicado pela diferença entre as competências exigidas dos discentes do ensino superior – conteúdos, habilidades e atitudes — em relação às exigidas no Ensino Médio. O tipo de estudo necessário, as atividades exigidas, o raciocínio demandado, entre várias outras características do trabalho acadêmico, são diferentes do esquema de memorização de conteúdos utilizado na maioria das escolas [22]. As dificuldades levam à frustração e ao desapontamento com o curso, o que provoca a também alta taxa de evasão do primeiro semestre. Conteúdos novos, como lógica de programação, que não são vistos no Ensino Básico, possuem os mesmos índices de reprovação das demais disciplinas, o que parece indicar que o problema é a localização do componente no primeiro semestre, quando os estudantes estão se habituando às exigências do ensino superior. Como não é possível eliminar a existência de conteúdos no primeiro semestre, outra solução é necessária.

A literatura sobre ensino de programação apresenta diversas abordagens para a solução dos problemas levantados. Entre essas abordagens estão jogos, ferramentas que promovem atividades interativas e robótica educacional, seja por meio do uso de computadores ou de atividades desplugadas [5]. Entre as ferramentas computacionais, destaca-se o uso de linguagens visuais para programação, em que ícones podem ser arrastados na tela para formar as partes executáveis de um programa. O apelo da programação visual tem sido usado como argumento para a obtenção de um aumento na motivação dos jovens aprendizes. Ainda assim, essa organização não elimina a questão de que o processo de composição dos blocos (i.e., programação) pode dar mais ênfase à sintaxe do que à semântica da linguagem, dependendo de como é o fluxo de execução e semântica subjacentes dos elementos sintáticos visuais.

O objetivo deste trabalho é apresentar um levantamento sobre as linguagens visuais atualmente usadas para ensino de programação. O método usado foi a revisão sistemática da literatura e a análise dos trabalhos foi centrada no paradigma de programação subjacente. O restante do texto está organizado como se segue: a Seção 2 descreve o arcabouço teórico e as relações existentes entre paradigmas de programação e as formas de pensar a construção de software, que motivam este trabalho; os trabalhos relacionados são descritos na Seção 3; a Seção 4 mostra o protocolo da revisão; a Seção 5 apresenta os trabalhos encontrados no processo; a Seção 6 sistematiza e discute os achados e a Seção 7 conclui o texto.

2 PARADIGMAS E COMPETÊNCIAS

O artigo *Computational Thinking* [76] marca a discussão atual sobre ensino de Computação, com cerca de 2.500 citações registradas

EduComp'22, Abril 24-29, 2022, Feira de Santana, Bahia, Brasil (On-line)

no site da ACM. O "pensamento computacional" é um conjunto de habilidades e atitudes com foco na solução de problemas, competência necessária para todos, independente de área de atuação. Em seu artigo, Wing estabelece diretrizes gerais para que o objetivo de "pensar computacionalmente" seja atingido. Posteriormente, Wing conceitua Pensamento Computacional como "os processos mentais envolvidos na formulação e na expressão da solução de um problema, de forma que tanto pessoas como máquinas possam resolvê-lo" (trad. nossa) [77]. A abstração é considerada a habilidade essencial para a mobilização desses processos, sendo decorrente do reconhecimento e definição de padrões, generalização a partir de instâncias concretas e parametrização para adaptação de soluções a contextos específicos.

O discurso sobre competências deve encontrar aporte nos procedimentos operacionais usados nos processos de ensino e aprendizagem. Conforme discutido em [12] a respeito dos termos *crítica* e *interdisciplinaridade*, não basta o entendimento do conceito de abstração: é preciso construir, aplicar, avaliar e sistematizar os resultados do desenvolvimento de competências relacionadas ao termo, a partir de operações bem definidas. Não é suficiente a declaração de que um/a método/técnica/linguagem dá suporte ao desenvolvimento da capacidade de abstração: é preciso que os processos envolvidos nesse desenvolvimento sejam explicitados.

O exercício da abstração é pouco explorado no Ensino Básico brasileiro. A Matemática, lugar de destaque do pensar abstrato e do reconhecimento de padrões para modelagem de fenômenos, é ensinada de forma puramente operacional. A ideia de Matemática como sinônimo de "fazer contas" e de "números" está enraizada graças a esse enfoque. Elementos de Álgebra Universal [9], que fundamentam a organização de dados e seus mapeamentos, somente são estudados no Ensino Superior, quase sempre de forma descontextualizada. A "falta de base" anteriormente mencionada de fato existe. Contudo, possui natureza bastante distinta da falta de domínio dos algoritmos numéricos vistos na escola.

Paradigmas de programação determinam como o programador estrutura a solução de um problema, a partir do modelo matemático que dá suporte às estruturas da linguagem. Linguagens podem implementar construções de um ou mais paradigmas e a facilidade de codificar uma solução deriva da consistência entre a forma de pensar a solução e as construções disponibilizadas [51]. A eficiência da solução também advém do projeto e implementação da linguagem. Por exemplo, a linguagem C permite desenvolver um programa funcional (tipado), embora a ausência de mecanismos de *garbage collection* comprometa a efetividade da solução; um programa concorrente em Java é menos eficiente do que uma solução em Erlang, que contém construções de concorrência nativas.

O modelo matemático subjacente a um paradigma estabelece quais são os seus elementos principais e as operações daí decorrentes. No paradigma imperativo, a memória determina o estado e o fluxo de execução de um programa. A atribuição é a única operação capaz de alterar esses elementos. Fortemente baseado na Arquitetura de von Neumann e nas primeiras linguagens de montagem, o modelo de execução imperativo permite efeitos colaterais na execução de funções/procedimentos/subrotinas por meio de atribuição de valores a variáveis globais ou a parâmetros passados por referência ou resultado [57]. Estruturas de encapsulamento de dados e código usualmente existem, mas não têm uso obrigatório, facilitando a escrita de programas pouco estruturados, com enormes blocos de códigos que executam muitas operações diferentes e não relacionadas em sequência. Esse modelo não orienta o aprendiz para o exercício de reconhecimento de padrões e parametrização de soluções, elementos essenciais da abstração.

O paradigma de orientação a objetos favorece o exercício da abstração, visto que a construção de classes envolve pensar sobre padrões, separar dados de suas funcionalidades e evidenciar oportunidades de parametrização de objetos e de seus métodos. Diferentemente do paradigma imperativo, a abstração necessária para construção de um modelo de classes/objetos exige o desenvolvimento de habilidade de modelagem, antes de pensar em como os métodos serão implementados. Contudo, o uso de linguagens orientadas a objeto com fluxo de execução imperativo acaba produzindo o mesmo tipo de problema descrito no parágrafo anterior, especialmente quando o foco na codificação (e não na modelagem) é privilegiado [22, 31, 43].

A falta do exercício da abstração tem consequências: a Engenharia de Software moderna preconiza a reutilização de especificações, modelos e código, com vistas ao aumento da produtividade das equipes de desenvolvimento e diminuição dos custos relacionados aos procedimentos de verificação, pelo uso de artefatos já testados, inclusive em ambiente de produção [46]. O reúso exige do desenvolvedor um pensamento voltado à composição de elementos conhecidos para a solução de um problema novo. A habilidade de compor especificações ou código não pode ser desenvolvida quando os estudantes, a cada nova tarefa, são instados a construir sempre uma solução a partir do zero.

Uma das formas de desenvolver habilidades de composição de artefatos de software é ensinar os alunos a programar com linguagens que tenham a composição como operação de destaque, exigindo o uso de componentes já desenvolvidos para a construção de uma nova solução [81]. O paradigma funcional possui essa característica [53]: construir um programa funcional monolítico é mais difícil do que compor especificações menores. Argumentos contrários ao uso do paradigma funcional baseiam-se na inadequação sintática ao ensino, com listas aninhadas na linguagens LISP [68] e modelos que misturam formalismos de funções com regras ou elementos de linguagens imperativas, como em ML [70], Haskell [66], Elixir [2] ou Scala [41]. Abordagens orientadas a objetos com fluxo de execução funcional, como CLOS [28], facilitam a programação de estruturas de dados complexas, mantendo as vantagens da composicionalidade. A estrutura de chamada de funções aninhadas, por sua vez, não favorece o pensar sequencial associado às linguagens textuais imperativas e o modelo de regras associado ao paradigma lógico é considerado difícil. O pensar em termos sequenciais é relevante, visto que os futuros programadores não só escrevem horizontalmente da esquerda para a direita, como deverão migrar para linguagens imperativas textuais, exigidas nos cursos de graduação e em atividades profissionais.

Representações visuais de programas são usadas desde os anos 1950 para a visualização do fluxo de controle de programas no paradigma imperativo, denominadas diagramas de fluxo ou fluxogramas. Nos anos 1960, diagramas de Nassi-Schneiderman (ou Chapin) foram desenvolvidos para eliminar a possibilidade de programação não estruturada com saltos não condicionais no código (i.e., "go

to"). Representações visuais são comumente usadas em Engenharia de Software, para especificação de componentes e funcionalidades. Exemplos são os diagramas da linguagem UML e os modelos ER. Formatos visuais são vantajosos por expressarem informação que pode ser facilmente entendida, mesmo por não especialistas.

Com a disseminação de melhores processadores e aceleradores gráficos, no final do Século XX, a programação visual passou a ser uma realidade. A apresentação de linguagens visuais costuma ser mais focada na sua usabilidade e menos no paradigma, ou modelo de execução, subjacente aos programas desenvolvidos nessas linguagens. Como a forma de resolver um problema computacional é diretamente ligada ao paradigma usado, a motivação deste trabalho é investigar o paradigma subjacente às linguagens visuais orientadas ao ensino de programação.

3 TRABALHOS RELACIONADOS

A revisão da literatura apresentada em [43] busca trabalhos sobre ensino de programação em múltiplas disciplinas que possam contribuir com professores de programação, discutindo as diferenças entre habilidade de resolução de problemas e uso de linguagens, paradigmas e metáforas de programação, ambientes e visualização da execução de programas. Não é uma revisão sistemática e não apresenta resultados quantitativos. O escopo é o ensino superior e não há menção a linguagens visuais. [58] apresenta objetivos similares de análise de trabalhos publicados em eventos de Educação em Computação no período 2005-2009, voltados ao ensino de programação, não necessariamente introdutória. O trabalho analisa a avaliação nos produtos desenvolvidos em vez da análise do processo e do entendimento das dificuldades enfrentadas. A conclusão é que existe a necessidade de integração de modelos de ensino e aprendizagem nos cursos de programação em vez de abordagens puramente quantitativas e locais.

[35] é o relatório do grupo de trabalho formado no âmbito da conferência ITiCSE da ACM, com objetivo de prover um panorama dos avanços no ensino de programação, entre 2003 e 2017. As questões de pesquisa dizem respeito aos principais temas, desenvolvimentos e evidências sobre o ensino de programação. O método foi o de revisão sistemática estruturada da literatura. Os trabalhos foram divididos em quatro grupos: estudantes, ensino, currículo e avaliação. Dos 5.056 artigos analisados, 53 tratavam de escolha de paradigmas de programação no ensino, sendo que a análise feita mostrou que a abordagem desse tema vem diminuindo ao longo dos anos. Linguagens visuais aparecem de relance no texto, com 8 trabalhos que trazem o termo no título listados na bibliografia. O relatório conclui que as áreas mais teóricas sobre o ensino e aprendizagem de programação encontram-se estagnadas, em detrimento de áreas relacionadas ao desenvolvimento de ferramentas e com maior foco no estudantes, atitudes, engajamento e questões de cunho social.

Outras revisões foram encontradas, todas com focos diferentes de linguagens visuais e seus paradigmas subjacentes. Alguns exemplos: [44] levanta e categoriza diversas ferramentas usadas no ensino introdutório de programação, incluindo algumas das presentes nesta revisão; [29] busca as metodologias usadas no ensino de programação, com questionamento sobre a relação entre as operações aplicadas e o desenvolvimento metodológico descrito nos trabalhos; [7] analisa 45 trabalhos publicados a partir de 2012, identificando as ferramentas usadas, o nível de ensino e a execução de atividades, voltado à avaliação dos trabalhos quanto à sua clareza e completude.

O trabalho de revisão mais próximo ao desenvolvido neste artigo é [62], que apresenta um mapeamento sistemático da literatura que busca entender os cenários em que a programação com blocos é utilizada no ensino e na aprendizagem do pensamento computacional na educação superior. O trabalho identifica as ferramentas usadas, tipos de avaliação, níveis de ensino e países de origem dos trabalhos. A busca mostra um crescimento de trabalhos envolvendo pensamento computacional. Tratando-se de um mapeamento, não há análise crítica sobre os trabalhos pesquisados e os resultados não aprofundam a questão das habilidades relacionadas ao pensamento computacional estarem efetivamente sendo desenvolvidas, questão que nem mesmo é levantada no trabalho. O número de trabalhos analisados (33) é restrito, limitando o escopo. Os achados sobre ferramentas e linguagens são compatíveis com as deste trabalho: Scratch foi a linguagem predominante, seguida pelo App Inventor.

Outros mapeamentos sistemáticos, relacionados ao pensamento computacional, mas sem relação com linguagens visuais e paradigmas, tais como [1] também foram encontrados. Quase todos apresentam trabalhos sem uma análise crítica detalhada sobre o uso e desenvolvimento efetivo das habilidades do pensamento computacional nas atividades propostas. Essa questão será aprofundada na discussão dos resultados do trabalho e na resposta às questões de pesquisa formuladas.

A pouca quantidade de revisões encontradas explica-se pela dificuldade de delimitar um tema no qual a quantidade de produções é grande mas sem os mesmos critérios e padrões. A discussão sobre paradigmas de linguagem, pujante durante as últimas décadas do Século XX ao ponto de ser considerada "religiosa" [61] vem decaindo [35], mas é necessária na medida em que novos paradigmas de bancos de dados não relacionais e de desenvolvimento de software passam a ser usados na indústria.

Nenhum trabalho avaliando paradigmas de linguagens visuais foi encontrado, assegurando-se o ineditismo desta abordagem, pelo menos quando restrita aos critérios de inclusão da revisão.

4 MATERIAL E MÉTODOS

O trabalho desenvolvido é de natureza bibliográfica e qualitativa. O método foi adaptado de [30] e [17], consistindo nas seguintes etapas: (*i*) definição das questões de pesquisa da revisão; (*ii*) definição das fontes de pesquisa; (*iii*) definição das *strings* de busca; (*iv*) filtragem dos resultados; (*v*) seleção dos trabalhos encontrados; (*vi*) revisão e análise dos trabalhos; (*vii*) incorporação de referências; e (*viii*) sistematização dos achados.

As questões de pesquisa da revisão da literatura são:

- **Q1** Quais são as linguagens visuais existentes para ensino de programação e que paradigma implementam?
- **Q2** Quais são os principais métodos/linguagens visuais de ensino de programação usados nos diferentes níveis de ensino?
- **Q3** Existem implementações de linguagens visuais que considerem os princípios do pensamento computacional?

As seguintes palavras-chave foram escolhidas para a busca do referencial usado neste trabalho:

Termo	Sinônimos
Ensino	ensino, aprendizado, método, estratégia, learning, teaching, method, strategy
Programação	programação, programa, algoritmo, programming, algorithm
Linguagem	ferramenta, linguagem, sistema, software, tool, language, system
Pensamento computacional	"pensamento computacional", "computational thinking"
Propriedades	composicional, modular, visual, compositional
Nível de ensino	"ensino fundamental", "ensino médio", "ensino básico", "high school", "middle school", "elementary school"

As *strings* consistem em uma conjunção de cláusulas com a disjunção dos sinônimos de cada termo em seu interior. Testes foram conduzidos para verificar os retornos, adaptando as strings para buscas mais efetivas. Os repositórios não têm formato padrão para busca e são sensíveis à quantidade de termos e operadores usados, sendo necessário adaptar as strings para cada repositórios. As strings usadas, usando os operadores lógicos de conjunção (AND), disjunção (OR) e negação (NOT), são:

- **S1** ("programming learning" OR "programming teaching" OR "programming tool" OR "programming language") AND (algorithm OR software) AND ("computational thinking") AND (visual OR compositional) AND ("high school" OR "middle school") AND NOT ("c programming"OR java OR reading)
- **S2** ("programming learning" OR "programming teaching" OR "programming tool" OR "programming language") AND (algorithm OR software) AND ("computational thinking") AND (visual OR compositional) AND NOT ("c programming"OR java OR reading)
- **S3** (ensino de programação *OR* aprendizado de programação) *AND* (ferramenta *OR* linguagem)
- **S4** ("programming learning" OR "programming teaching") AND ("visual programming language" OR "visual language" OR blocks) AND "computational thinking"

As fontes de referências bibliográficas foram os repositórios ACM Digital Library (https://dl.acm.org/), IEEE Xplorer (https://ieeexplore.ieee.org/), Wiley Online Library (https://onlinelibrary.wiley.com/), SBC Open Libray (https://sol.sbc.org.br/index.php/wei) e Revista Brasileira de Informática na Educação (https://www.br-ie.org/pub/index.php/rbie). A sintaxe da plataforma IEEE Xplorer é sensível à ordem dos operadores e espaços em branco na *string* fazem com que os demais termos seja ignorado. Essa questão não é documentada no manual de uso e gerou resultados inconsistentes antes que o problema fosse descoberto. Na mesma plataforma, a busca avançada produziu resultados fora do escopo desejado; a busca por linha de comando foi então usada. A biblioteca SOL não permite buscas globais e as buscas foram restritas ao Workshop sobre Ensino de Computação (WEI) e Workshop sobre Informática na Escola (WIE). Na plataforma Wiley Online Library o filtro de busca pelos assuntos *Education* e *Computer Science* foi usado. O repositório Google Scholar (https://scholar.google.com) foi descartado pela falta de consistência no retorno dos resultados da busca.

As strings de busca exigem que o termo "pensamento computacional" apareça no título, resumo ou texto completo do trabalho. A restrição diminuiu o número de artigos e foi feita por (i) estarmos interessados em trabalhos sobre o referencial atual da área e o pensamento computacional, com a devida crítica necessária, é um referencial importante sobre formação em Computação; (ii) o protocolo de revisão prever a incorporação de trabalhos relevantes que apareçam no referencial bibliográfico dos artigos. Dessa forma, trabalhos que tenham escapado nas buscas devem aparecer no resultado final.

Os critérios de inclusão são: (i) propostas de linguagens visuais para o ensino de programação em qualquer fase de ensino, (ii) discussão do ensino-aprendizagem dentro do arcabouço do pensamento computacional, (iii) análise de resultados educacionais discutidos a partir trabalho realizado. Os critérios de exclusão são: (i) trabalhos que usem linguagens textuais, (ii) trabalhos anteriores a 2016, a menos que tenham sido referenciados em algum dos trabalhos incluídos e que respondam a pelo menos uma das questões de pesquisa da revisão.

A continuidade deu-se pela leitura dos resumos, introduções, conclusões e verificação das referências dos trabalhos selecionados. Nessa fase foram identificados os potenciais trabalhos relevantes que não foram encontrados pelos métodos de pesquisa utilizados. A Tabela 1 sumariza os resultados da aplicação do protocolo.

Tabela 1: Síntese dos achados no processo de busca

Repositório	String	Local	Total	Incl.
ACM Digital Library	S1 S2	Título Resumo	11	5
IEEE Xplorer	S1 S2	Título Resumo	66	8
Wiley Online Library	S4	Texto	53	8
SOL (WEI)	S3	Texto	57	14
SOL (WIE)	S3	Texto	53	19
ACM Transactions on Computing Education	S1	Texto	78	4
International Computing Education Research Workshop	S1	Texto	52	2
Information Technology Education Conference	S1	Texto	25	1
SIGCSE: Computer Science Education	S1	Texto	35	2
Revista Brasileira de Informática na Educação	S3	Título	5	1
Referencial bibliográfico	–	–	–	12
Total			**435**	**76**

Ao final do processo, foi realizada uma análise em abrangência e profundidade dos trabalhos selecionados. Na análise dos trabalhos, novas fontes puderam ser identificadas e foram incorporadas à revisão. O processo de busca terminou quando nenhum novo trabalho relacionado foi encontrado no processo.

5 LINGUAGENS VISUAIS PARA ENSINO DE PROGRAMAÇÃO

A linguagem Scratch (scratch.mit.edu/statistics), ferramenta do MIT com sintaxe baseada em blocos, aparece como destaque nos trabalhos desta revisão por conta do tamanho da sua comunidade de usuários. Conceitualmente, a linguagem Scratch é composta por

blocos que representam diferentes instruções, que podem ser combinados e posteriormente executados por um personagem animado. Os diferentes tipos de instrução (movimento, aparência, controle, som, eventos, ambiente, operadores, variáveis) são caracterizados por cores e formatos distintos. Um programa em Scratch é construído pela concatenação de blocos representando instruções. As funções podem ser organizadas com novos blocos definidos pelo usuário, o que permite a modularização (parcial) dos programas.

Figura 1: Função fatorial em Scratch

A Figura 1 mostra a função de cálculo do fatorial de um número como um bloco Scratch. O fluxo de execução das instruções e sua organização seguem o paradigma imperativo, com variáveis globais cujo valor é mostrado na tela. É interessante notar que os módulos de ajuda não mencionam a construção de blocos, justamente a parte da criação de elementos que podem ser compostos em alto nível. Funções não podem ser implementadas, visto que não há possibilidade de geração de valores de retorno. Os programas em Scratch seguem uma organização de código vertical e monolítica, similar à usada em fluxogramas ou (fracamente) diagramas de Nassi-Schneiderman. A notação Scratch parece ter sido inspirada nesses últimos.

A linguagem Scratch serviu de inspiração para outras linguagens/ferramentas desenvolvidas com foco em ensino de programação. BEESM (*Block-based End-user programming tool for Smart Environments*) [56] é uma ferramenta de programação visual em ambiente web que pode ser utilizada para programação de robôs, microcontroladores e ambientes inteligentes. A linguagem é organizada em blocos similares aos de Scratch. Os programas construídos podem ser automaticamente traduzidos e visualizados em código PHP ou na linguagem de programação do microcontrolador Arduino. Segundo os autores, a utilização de blocos reduz os erros de sintaxe e facilita a manipulação das estruturas do código.

O ambiente de programação visual em blocos da BEESM foi construído utilizando Google Blockly [42] , uma biblioteca de código aberto para adicionar código baseado em blocos em uma aplicação. O Google Blockly facilita a construção de linguagens em blocos, pois fornece uma gramática e uma representação para programação que os desenvolvedores podem usar em seus aplicativos. Os blocos disponibilizados pela Google Blockly são visualmente semelhantes aos blocos do Scratch.

O *App Inventor*, ferramenta originalmente lançada pela Google e atualmente mantida pelo MIT, serve para o desenvolvimento de aplicações para dispositivos móveis que façam uso do sistema operacional Android. O *App Inventor* usa blocos para a programação de funcionalidades específicas de dispositivos móveis, com uma tela para construção da parte visual da aplicação, permitindo a criação de aplicativos por iniciantes, com poucos conhecimentos sobre linguagens e ambientes de programação. A Figura 2 apresenta a função fatorial definida no ambiente do *App Inventor*.

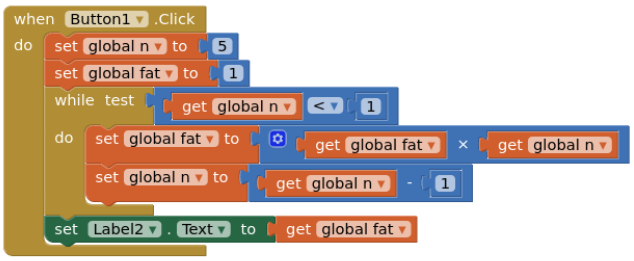


Figura 2: Função fatorial no *App Inventor*

Além da BEESM e do App Inventor pode-se mencionar uma série de soluções que utilizam o Scratch para a programação de dispositivos com propósito educacional como Arduino, Lego Mindstorms e Raspberry Pi. Dentre esses trabalhos encontram-se as ferramentas Make Code [18], Scratch4Arduino [24] e Modkit [37] que tornam possível a utilização do Scratch para a programação com Arduino e as linguagens DuinoBlocks for Kids [48], Lofi Blocks [54] e Sucre4Kids [69] que possuem estruturas de blocos coloridos semelhantes ao Scratch e também são voltadas para a programação de placas Arduino. A ferramenta Enchanting [36] também utiliza o Scratch, mas para programação de Lego Mindstorms NXT. A estrutura da linguagem utilizada por essas ferramentas é a mesma do Scratch, apenas adicionando blocos específicos para a programação do dispositivo em questão.

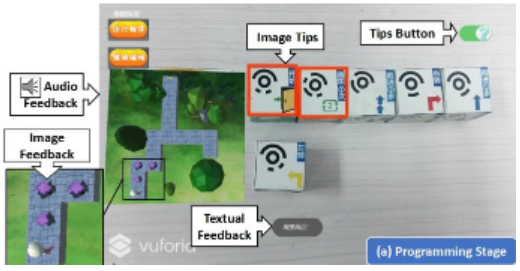
Outra ferramenta utilizada em alguns trabalhos foi a plataforma Hora do Código do site Code.org, que consiste em diversos jogos e tutoriais temáticos que utilizam principalmente a linguagem Scratch e linguagens baseadas em Scratch com o objetivo de ensinar programação para crianças. Outra aplicação da linguagem Scratch observada em alguns trabalhos foi o uso em atividades voltadas para alunos de Ensino Fundamental e Médio, com o objetivo de trabalhar conceitos relacionados às disciplinas de Matemática e Ciências, bem como em atividades para o incentivo do pensamento computacional e das habilidades relacionadas.

Conceitualmente, todas as abordagens com base em Scratch possuem as mesmas caraterísticas de programação vertical com fluxo de execução imperativo.

Os *kits* de robótica da Lego™ [45], nomeadamente Mindstorms EV3 [33] e Education WeDo [32], são ferramentas utilizadas no ensino de programação, principalmente com foco em alunos do Ensino Fundamental e Médio, mas com propósito de movimentar aparatos robóticos com vistas à execução de tarefas. Até 2019, ambos os kits utilizavam a linguagem de blocos feita com LabView [39], composta por blocos quadrados conectados na horizontal representando as funções que devem ser executadas pelos componentes do kit (motores, sensores e tela). O kit Lego Education WeDo é direcionado para

alunos das séries iniciais do Ensino Fundamental e possui blocos mais coloridos, símbolos maiores e com menos detalhes em relação à linguagem utilizada pelo kit Mindstorms EV3, o que facilita a diferenciação dos blocos. Já o kit Mindstorms EV3 (bem como seu antecessor NXT) possui blocos com mais informações, permitindo alterações como direção ou número de giros de um motor. A partir de 2019, o kit Mindstorms EV3 passou por uma alteração no software da ferramenta e começou a utilizar a linguagem Scratch para a programação dos projetos. Apesar das diferenças entre a organização do programas em ambas as linguagens, é possível construir códigos equivalentes aos desenvolvidos com a linguagem anterior de uma forma bem direta. Os kits Lego também seguem o esquema de fluxo de execução imperativo, com programas monolíticos e variáveis globais.

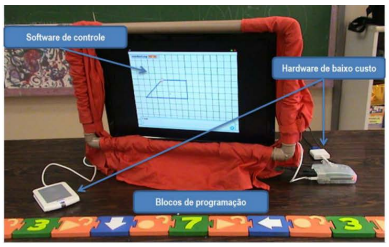
Várias outras iniciativas buscam criar ferramentas e linguagens acessíveis para diversos níveis de ensino, diferentes do esquema de blocos monolíticos do Scratch. Uma tendência observada na literatura recente é o desenvolvimento das chamadas ferramentas tangíveis [26], que usam objetos físicos para representar elementos de linguagens de programação como comandos e estruturas de fluxo de controle em vez de utilizar imagens e palavras na tela do computador, principalmente visando alunos das séries iniciais do Ensino Fundamental e da Educação Infantil. Ferramentas como Blocos Tangíveis [25], ARCat [16] e a Ar-maze [27] (Figura 3), Cubetto [10], TaPrEC [11] (Figura 4) e Scottie Go! [72] aliam artefatos tangíveis e realidade aumentada como forma de tornar a ferramenta acessível a estudantes mais jovens e facilitar o trabalho em equipe. Os softwares mencionados leem e interpretam os códigos construídos pelos usuários por meio da câmera de um *smartphone* ou *tablet* e executam o algoritmo utilizando realidade aumentada. Esse tipo de ferramenta tem potencial para aumento da motivação dos alunos em aprender programação, além de eliminar a necessidade das escolas de possuírem laboratórios de informática para essa finalidade, visto a ubiquidade dos dispositivos móveis na sociedade. Ainda que nem todos possuam esses equipamentos, um bom *smartphone* hoje possui desempenho similar aos *desktops*, custando menos da metade do preço. A diferença deve diminuir nos próximos anos, seguindo a tendência atual e a quantidade de equipamentos vendidos.



Fonte: [27]

Figura 3: Imagem da ferramenta Ar-maze

Outro aspecto observado foi o formato dos blocos utilizados em cada linguagem. Ar-maze usa pequenos cubos de madeira que podem ser conectados com ímãs. Cada bloco representa um tipo de movimento que o personagem do software pode realizar, além de marcações para início e fim do programa e instruções de laços. A Figura 3 apresenta uma imagem do ambiente de programação Armaze. A programação em Cubetto usa peças no formato de setas que são encaixadas em um tabuleiro como instruções sequenciais para movimentar um robô no formato de um cubo de madeira. TaPreEC (Figura 4) usa blocos de madeira coloridos com uma etiqueta RFID no verso que são encaixados em uma sequência horizontal; uma placa Raspberry Pi que executa o programa e mostra o resultado em um monitor. O jogo Scottie Go! possui um conjunto de cartões coloridos com instruções que são encaixados em uma sequência vertical e escaneados por uma aplicação móvel que mostra a execução dos comandos por um personagem.

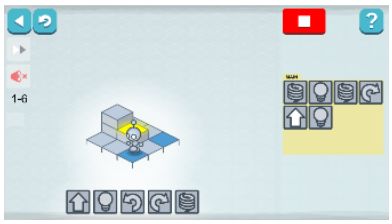


Fonte: [11]

Figura 4: Imagem da ferramenta TaPrEC

Jogos têm sido usados para o ensino de programação, especialmente pelo seu apelo lúdico. Uma tendência observada foi a organização dos jogos em fases, em que cada fase representa um labirinto ou quebra-cabeça e o usuário precisa construir um conjunto de instruções para guiar o personagem para o fim do labirinto ou para o objetivo da missão. Exemplos desses jogos são Lightbot [63] (Figura 5), Robotizen [14], Move [67], NoBug's Snack Bar, VR-OCKS, Jogo RPG de Sarkar [52], e ELIS [65]. O jogo Lightbot possui blocos quadrados que controlam os movimentos do personagem e as ações de acender e apagar casas do tabuleiro. O código é construído em sequência horizontal. A complexidade dos níveis aumenta gradativamente desbloqueando, conforme o avanço do jogador, novos blocos para a construção de procedimentos – que são desenvolvidos em outro espaço e chamados na área principal por meio de um bloco específico – e de *loops* (ciclos) iterativos. Esses *loops* são executados até que o nível seja concluído corretamente ou que o jogador interrompa a execução. O jogo Robotizen possui estrutura semelhante ao Lightbot, com blocos quadrados em sequência horizontal que controlam movimentos e ações do personagem, bem como estruturas de decisão e repetição. O jogo Move possui blocos retangulares que são organizados de forma vertical e controlam a movimentação do personagem, por meio de estruturas de repetição e de decisão. O jogo Elis possui blocos retangulares que controlam a movimentação e as ações de pegar e soltar o lixo, que deve ser descartado no local correto. O jogo RPG é composto por blocos quadrados com comandos representados por símbolos simples alinhados de forma sequencial que controlam a movimentação do personagem, estruturas de condição e loops.

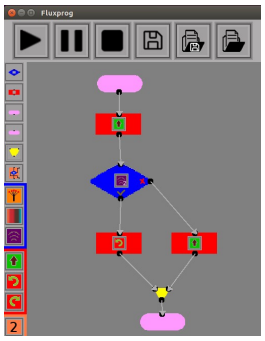
A forma mais tradicional de ensino de programação nas primeiras décadas da História da programação de computadores também foi migrada para ferramentas de ensino: Fluxprog [19] e Modelix [38] são linguagens baseadas em fluxogramas. Fluxprog (Figura 6)



Fonte: www.lightbot.com

Figura 5: Imagem do Jogo LightBot

possibilita a programação de dispositivos construídos com Arduino e de robôs em um simulador virtual 3D. Os blocos utilizados no fluxograma possuem uma diferenciação de cor para cada tipo de comando: início e fim do programa, ciclo, ações, testes condicionais e fusão de fluxo de controle. Modelix também tem foco em robótica educacional, permitindo a programação de placas Arduino.

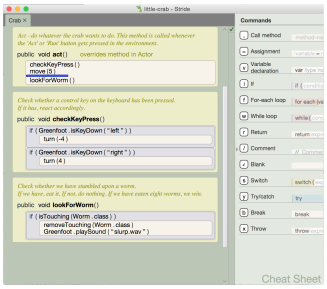


Fonte: [19]

Figura 6: Imagem da ferramenta FluxProg

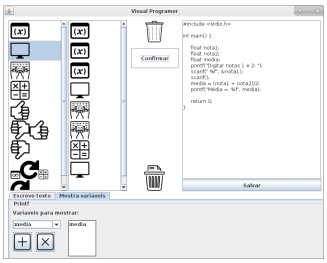
Outro modelo de linguagem visual encontrado na literatura é o da linguagem Stride [47] (Figura 7), que usa uma parte textual oriunda da linguagem Java, com adição de quadros (*frames*) para o ensino de programação. Os quadros são semelhantes aos blocos, organizados de forma vertical, com espaços definidos para inserção de comandos textuais. A linguagem Stride visa reduzir erros de sintaxe cometidos pelo desenvolvedor iniciante. Foram realizadas atividades com alunos do Ensino Fundamental com um grupo de alunos utilizando Java e outro utilizando Stride e, segundo o autor, o grupo trabalhando com Stride progrediu mais rápido na execução da atividade, passando menos tempo editando a sintaxe. Isso sugere que a edição baseada em quadros é uma ferramenta útil para reduzir a carga cognitiva relacionada à sintaxe para programadores iniciantes e pode levar a uma melhora na performance de tarefas em programação.

A linguagem Visual Programmer (Figura 8) tem foco no ensino de programação para alunos surdos e utiliza símbolos para representação das instruções. A ferramenta permite que o código simbólico seja convertido para linguagem C possibilitando que o aluno surdo compreenda a aplicabilidade de cada instrução dentro do código-fonte [3].



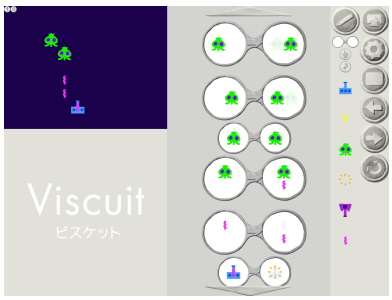
Fonte: [47]

Figura 7: Código na linguagem Stride



Fonte: [3]

Figura 8: Imagem da ferramenta Visual Programmer



Fonte: [74]

Figura 9: Imagem da ferramenta Viscuit

A linguagem que mostrou um padrão diferente do observado em outros trabalhos foi Viscuit [74] (Figura 9), linguagem de programação visual voltada para alunos da Educação Infantil. Por ter como usuários crianças não alfabetizadas, Viscuit não possui informações textuais e é composta somente por imagens. A linguagem segue um sistema de reescrita: o código é construído com regras representadas por uma estrutura semelhante a um óculos (dois círculos conectados). A criança desenha as imagens a serem utilizadas ao longo do código. Quando o código é executado, a imagem do círculo à esquerda (lado esquerdo da regra) é substituída pela imagem do círculo à direita (lado direito da regra). É possível também utilizar imagens prontas que representam comandos de clicar ou executar sons. A programação baseada em regras é fortemente associado ao paradigma lógico de programação. A inexistência de variáveis em Viscuit, contudo, afasta o modelo de execução deste paradigma,

ainda que o conceito de unificação esteja presente no modelo de execução da linguagem.

6 RESULTADOS E DISCUSSÃO

6.1 Questões de pesquisa

Os dados produzidos em uma revisão sistemática da literatura podem ser analisados sob diferentes perspectivas. O objetivo desde trabalho foi analisar os resultados sob a ótica dos paradigmas de programação que determinam o fluxo de controle das linguagens visuais voltadas ao ensino de programação. Juntamente, foi feita uma análise sobre o tema pensamento computacional, citado nos trabalhos, em sua efetiva utilização no ferramental analisado e o nível de ensino em que a linguagem foi utilizada. A organização sintática dos elementos visuais foi considerada, mesmo que não altere o paradigma da linguagem, por ter impacto na transição do programador para linguagens textuais. A seguir, os achados são analisados à luz das questões de pesquisa, apresentadas na Seção 4.

Questão de pesquisa Q1 - Quais são as linguagens visuais existentes para ensino de programação e que paradigma implementam?

A Tabela 2 apresenta uma síntese da revisão realizada, em relação à questão Q1. Na primeira coluna é indicado o paradigma de programação seguido, na segunda coluna a estrutura sintática visual utilizada, a terceira coluna mostra a organização dos elementos sintáticos, a quarta coluna apresenta o agrupamento das linguagens que fazem uso de cada combinação dos elementos das colunas anteriores e, na última coluna, o número de trabalhos encontrados sobre as linguagens da coluna anterior. O agrupamento das linguagens e ferramentas encontradas a partir de seu paradigma e estrutura facilita o entendimento e a análise posterior. Note-se que alguns trabalhos traziam mais do que uma linguagem, o que fez com que o número de vezes que uma linguagem aparece em um trabalho possa ser maior do que um.

Do ponto de vista do paradigma de programação usado, não há dúvidas de que a quase totalidade das linguagens analisadas segue o paradigma imperativo. Esse resultado é compreensível, pela dominância do paradigma na indústria e na academia. A surpresa veio da total ausência de discussão, nos textos analisados, sobre os paradigmas subjacentes a cada linguagem e seu impacto no aprendizado. Modelos de aprendizagem são discutidos em poucos trabalhos e existem muito poucas análises quantitativas e metodologicamente fundamentadas sobre a efetividade das abordagens usadas.

Questão de pesquisa Q2 - Quais são os principais métodos/linguagens visuais de ensino de programação usados nos diferentes níveis de ensino?

A linguagem Scratch é dominante: dos trabalhos analisados, 33 utilizaram Scratch e outros 22 trabalhos utilizaram linguagens em blocos com estruturas bastante semelhantes ao Scratch (Lego e outros).

As estruturas sintáticas da maior parte das linguagens analisadas consistem de blocos predominantemente quadrados, organizados de forma vertical ou horizontal. O uso horizontal aparece com mais frequência em linguagens voltadas para um público de menor idade, com uso de símbolos maiores e mais simples para a representação das instruções. A Figura 10 apresenta um exemplo de código na linguagem Scratch Jr., versão do Scratch para crianças. Dos trabalhos analisados, 19 utilizaram ferramentas com esse padrão, dentre elas os kits Lego, Scratch Jr., as ferramentas tangíveis Ar-maze, TaPrEC e Cubetto e os jogos Lightbot, Robotizen e o jogo RPG de [52].

A organização vertical, por outro lado, é mais comum em linguagens voltadas para estudantes do Ensino Médio e do Ensino Superior, como FluxProg, Modelix, Visual Programmer, VR-OCKS e Stride. Viscuit também uma uma organização de regras verticais e é a exceção, não só no paradigma usado. A organização das linguagens de programação textuais também é vertical, diferentes da forma como escrevemos e lemos um texto, na horizontal. A diferença na organização do fluxo de leitura/execução pode ter impacto na adaptação cognitiva dos aprendizes. Não encontramos trabalhos que investiguem essa questão, mas parece ser algo interessante.

Figura 10: Blocos da ferramenta Scratch Jr. como exemplo de uma linguagem com organização de blocos horizontal

Linguagens visuais que operam como linguagens intermediárias de domínio específico (ou *domain-specific languages*) [20], gerando código para aplicações móveis, microcontroladores ou dispositivos robóticos, foram encontradas em quantidade significativa (13 trabalhos) voltadas aos níveis Médio e Superior. A plataforma Arduíno foi o destino da maior parte das aplicações.

Sobre a estrutura sintática, 55 trabalhos utilizaram linguagens com organização de blocos vertical, 19 trabalhos utilizaram linguagens com organização de blocos horizontal e que as exceções foram seis trabalhos que utilizaram linguagens tendo como principais características fluxogramas, quadros, símbolos, regras ou uma combinação da organização de blocos verticais e horizontais. VR-OCKS utiliza um misto dos dois padrões, com organização de blocos na horizontal na maior parte das instruções, mas utilizando uma abordagem de organização de blocos na vertical quando se trata de estruturas de repetição.

Questão de pesquisa Q3 - Existem implementações de linguagens visuais que considerem os princípios do pensamento computacional?

A questão de pesquisa relacionada ao pensamento computacional, que orientou o procedimento de busca no protocolo de revisão, merece um tratamento à parte. Apesar de todos os trabalhos pesquisados nesta revisão mencionarem o arcabouço do pensamento computacional e muitos o usarem como justificativa para o uso das linguagens/ferramentas descritas, não foi possível responder à questão de forma geral. Somente dezoito dos trabalhos analisados apresentaram alguma justificativa para o uso da linguagem/ferramenta considerando o pensamento computacional, ainda que todos tenham o termo em seu corpo. Cinco trabalhos abordaram a contribuição das ferramentas Scratch e Lego Education WeDo nessa questão. Os trabalhos de [80] e [21] utilizam como base o

Tabela 2: Síntese dos resultados

Paradigma	Elementos	Organização	Linguagens	nº
Imperativo	Blocos	Vertical	Alice [13], *App Inventor* [79], BEESM [56], Blockly [42], DuinoBlocks for Kids [48], Enchanting [36], ELIS [65], Lofi Blocks [54], Make Code [18], Modkit [37], Move [67], NoBug's SnackBar [71], Pencil Code [4], ROBOTC Graphical [78], Scratch [49], Scratch4Arduino [24], SimBA-PLT [64], Snap! [50], Stencyl [34], Sucre4Kids [69]	60
		Horizontal	Education WeDo [32], Jogo RPG [52], Lightbot [23], Mindstorms EV3 [33], Robotizen [14], Scratch Jr. [6]	16
		Horizontal + Vertical	VR-OCKS [55]	1
	Blocos+Texto	Vertical	Stride [47]	1
	Diagramas	Vertical	Fluxprog [19], Modelix [38]	2
	Símbolos	Vertical	Visual Programmer [3]	1
	Tangível	Vertical	ARCat [16], Blocos Tangíveis [25], Scottie Go! [72]	3
		Horizontal	Ar-maze [27], Cubetto [10], TaPrEC [11]	4
Lógico	Regras	Regras	Viscuit [74]	1

framework e as dimensões-chave do pensamento computacional apresentadas em [8]. Em relação ao Scratch, com base em testes aplicados antes e depois de atividades realizadas com alunos do Ensino Fundamental e Médio, [80] relata um crescimento das habilidades de pensamento algorítmico e pensamento crítico nos alunos. Já em [73] os resultados mostraram uma aumento das habilidades de programação, resolução de problemas e criatividade. No entanto, qualquer tipo de ensino de programação vai promover um aumento de conhecimento sobre o que foi estudado. Esse tipo de conclusão é, essencialmente, tautológica. O trabalho de [45] também exemplifica a possibilidade do uso do kit Lego Education WeDo promover o pensamento computacional. Com base em um questionário respondido pelos alunos que usaram a ferramenta foi concluído que o kit auxiliou os alunos a pensar de forma criativa e lógica na resolução dos problemas apresentados. Questionários feitos com participantes de atividades quase nunca trazem resultados negativos. Esse tipo de viés de pesquisa é conhecido [75] e não é suficientemente discutido no trabalho.

A razão da impossibilidade da resposta está sumarizada nos exemplos dados acima: não basta declarar que uma linguagem ou abordagem de ensino desenvolve as habilidades associadas à abstração para que isso aconteça. Linguagens cujos programas não podem ser parametrizados, por exemplo, não permitem atingir uma das habilidades essenciais descritas por Wing em seus trabalhos. A abstração não está ligada necessariamente à codificação de um programa para executar um função, mas ao desenvolvimento de um algoritmo tão genérico quanto possível, parametrizado, para resolver uma classe de problemas ou múltiplas instâncias de um mesmo problema. O programa construído deve poder ser composto com outros programas para resolver problemas mais complexos. Linguagens que não implementam essas funcionalidades não podem declarar que fomentam o pensamento computacional.

Optamos por manter a questão de pesquisa, mesmo sem poder ser respondida, pois um resultado negativo em ciência é importante e aponta que mais estudos são necessários.

6.2 Impactos e limitações

O ensino de programação mantém-se um problema atual de pesquisa, mesmo com o vasto corpo de conhecimento existente na literatura. Nossa percepção é que abordagens que não considerem os problemas intrínsecos ao paradigma imperativo terão resultado inevitavelmente limitados. A construção de linguagens visuais voltadas ao ensino introdutório, nesse paradigma, tem o potencial de criar vícios iniciais de programação. Nenhum dos trabalhos analisados discute paradigmas. A inferência sobre o paradigma utilizado foi feita a partir dos programas construídos, das construções da linguagem e dos modelos de fluxo de dados e de controle implementados. Essa inferência não foi difícil de fazer, haja vista que as construções de linguagem puderam ser facilmente mapeadas para comandos de entrada/saída, atribuição, testes condicionais e laços iterativos. De todas as linguagens analisadas, somente uma – que usa um modelo de reescrita baseado em regras – não utiliza o esse paradigma. Eventuais erros de julgamento podem ter ocorrido. Contudo, foi possível criar programas e executá-los nas linguagens/ferramentas estudadas. A margem de erro, frente ao volume de trabalhos que usam um modelo imperativo conhecido, é muito pequena.

Além da questão dos paradigmas, outras análises são possíveis. A literatura menciona a existência e o uso de linguagens de programação visual voltadas para diversas faixas etárias. As análises mostram que linguagens visuais podem ser utilizadas com sucesso em diferentes níveis de ensino. Aplicações interdisciplinares no Ensino Fundamental e no Ensino Médio, em trabalhos onde a programação era utilizada como um meio para o aluno compreender conteúdos das áreas da Matemática ou Ciências, também foram encontrados e são iniciativas promissoras no contexto da Educação em Computação em particular e da Educação em Ciências de forma geral.

O protocolo apresentado tem o viés de privilegiar buscas em repositórios online, sendo a maior parte repositórios localizados nos Estados Unidos da América. No Brasil, privilegiou-se um repositório que sabe-se de antemão não conter a maioria dos trabalhos na área. Sabe-se que vários trabalhos podem ter ficado de fora dessa revisão por causa disso. Para remediar a questão, usou-se a estratégia de incorporação das referências importantes no protocolo.

7 CONCLUSÃO

O aprendizado de programação requer estratégias que permitam o desenvolvimento das habilidades de solução de problemas e sua codificação. Para tanto, as linguagens usadas devem favorecer a construção de competências relacionadas ao pensamento computacional, com foco na solução de problemas e não na retirada de erros de compilação.

Este trabalho investigou as linguagens visuais utilizadas para ensino de programação. O foco da análise foi o paradigma de programação subjacente, que determina o formato das estratégias usadas para solução de problemas e a estrutura das soluções. A quase totalidade das linguagens visuais é fundada no paradigma imperativo, também dominante em linguagens textuais de propósito geral.

Linguagens visuais de programação podem diminuir a carga cognitiva existente no processo de aprendizado. Linguagens que favoreçam o desenvolvimento de habilidades de abstração e composição de software podem ser mais efetivas na formação de programadores. Aprendizes que iniciaram com paradigmas mais apropriados à resolução de problemas, costumam levar o hábito da estruturação e composicionalidade para as linguagens do paradigma imperativo, que também permitem a estruturação de código. Assim, é surpreendente que linguagens com propósito didático escolham o paradigma imperativo como base.

Como trabalhos futuros, temos duas linhas em vista. A primeira é o desenvolvimento de uma linguagem visual funcional para ensino de programação [59, 60], que inclui organização horizontal e definição sequencial de funções, em vez de chamadas aninhadas, que são menos intuitivas. A razão de usar elementos mais familiares aos alunos é proveniente de achados das ciências cognitivas, que apontam que nosso cérebro tem dificuldades de reter conhecimento se não consegue ancorá-lo em conhecimentos prévios. A segunda versa sobre pensamento computacional e sua relação com as linguagens estudadas. O estudo não foi conduzido pela falta de mapeamentos entre a programação nas linguagens/ferramentas e as habilidades do pensamento computacional. Entendemos a necessidade de um estudo com maior profundidade sobre essa questão, pois mecanismos de desenvolvimento e mensuração da capacidade de abstração são necessários.

REFERÊNCIAS

[1] Friday Joseph Agbo, Solomon Sunday Oyelere, Jarkko Suhonen, and Sunday Adewumi. 2019. A Systematic Review of Computational Thinking Approach for Programming Education in Higher Education Institutions. In *Koli Calling '19: Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. 12 (1–10).

[2] Ulisses Almeida. 2018. *Learn Functional Programming with Elixir – New Foundations for a New World*. The Pragmatic Bookshelf. 198 pages.

[3] Gil Andrade, Diego Hoss, Ana Barbosa, and Lana Gomes. 2019. Metodologia Didático Simbólica como Alternativa para o Ensino de Programação de Computadores a Alunos Surdos. In *Anais do XXVII Workshop sobre Educação em Computação* (Belém). SBC, Porto Alegre, RS, Brasil, 473–482.

[4] David Bau and David Anthony Bau. 2014. A Preview of Pencil Code: A Tool for Developing Mastery of Programming *(PROMOTO '14)*. Association for Computing Machinery, New York, NY, USA, 21–24.

[5] Tim Bell, Ian H. Witten, and Mike Fellows. 2015. *Computer Science Unplugged* (3 ed.). Google Inc. 243 pages.

[6] M.U. Bers and M. Resnick. 2015. *The Official ScratchJr Book: Help Your Kids Learn to Code*. No Starch Press. https://books.google.com.br/books?id=Cw3SCgAAQBAJ

[7] R. P. Borges, P. R. F. Oliveira, R. G. da R. Lima, and R. W. de Lima. 2018. A Systematic Review of Literature on Methodologies, Practices, and Tools for Programming Teaching. In *IEEE Latin America Transactions*.

[8] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *In AERA 2012*.

[9] S. Burris and H.P. Sankappanavar. 1981. *A Course in Universal Algebra*. Springer, Berlin, Heidelberg. 276 pages.

[10] Lucía Gabriela Caguana Anzoátegui, María Isabel Alves Rodrigues Pereira, and Monica del Carmen Solís Jarrín. 2017. Cubetto for preschoolers: Computer programming code to code. In *2017 International Symposium on Computers in Education (SIIE)*. 1–5.

[11] Marleny Carbajal and M. Baranauskas. 2018. Programação tangível e construção de significado: criação de símbolos para o ambiente TaPrEC junto com professoras de ensino fundamental. In *Anais do XXIV Workshop de Informática na Escola* (Fortaleza, CE). SBC, Porto Alegre, RS, Brasil, 323–332.

[12] José Sérgio Carvalho. 2001. O discurso pedagógico das diretrizes curriculares nacionais: competência crítica e interdisciplinaridade. *Cadernos de Pesquisa* 12, 155–165.

[13] Matthew J. Conway. 1997. *Alice: Easy-to-Learn 3D Scripting for Novices*. Ph.D. Dissertation. Faculty of the School of Engineering and Applied Science at the University of Virginia.

[14] Igor Dantas, Jose Neto, Lucas Silva, Lourival Neto, Douglas Lima, Pasqueline Scaico, and Thaíse Costa. 2019. Ensino de lógica de programação no ensino fundamental utilizando o jogo Robotizen: um relato de experiência. In *Anais do XXVII Workshop sobre Educação em Computação* (Belém). SBC, Porto Alegre, RS, Brasil, 51–60.

[15] DEED/INEP. 2020. *Notas Estatísticas do Censo da Educação Superior 2019*. Technical Report. Brasília, DF. 32 pages.

[16] Xiaozhou Deng, Danli Wang, Qiao Jin, and Fang Sun. 2019. ARCat: A Tangible Programming Tool for DFS Algorithm Teaching *(IDC '19)*. Association for Computing Machinery, New York, NY, USA, 533–537.

[17] Diego Dermeval and Jorge A. P. M. Coelhoand Ig I. Bittencourt. 2020. *Mapeamento Sistemático e Revisão Sistemática da Literatura em Informática na Educação*. Vol. 2. CEIE/SBC, Chapter 3, 1–26.

[18] James Devine, Joe Finney, Peli de Halleux, Michał Moskal, Thomas Ball, and Steve Hodges. 2018. MakeCode and CODAL: Intuitive and Efficient Embedded Systems Programming for Education. *SIGPLAN Not.* 53, 6, 19–30.

[19] J. A. Fabro, E. Teixeira Paula, Á. F. G. P. Dias, and L. E. Skora. 2019. Programming Teaching Using Flowcharts in a Simulated Environment Focused on Introducing Practical OBR. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*.

[20] Martin Fowler. 2010. *Domain Specific Languages*. Addison-Wesley Professional. 640 pages.

[21] Ilenia Fronza, Nabil El Ioini, and Luis Corral. 2017. Teaching Computational Thinking Using Agile Software Engineering Methods: A Framework for Middle Schools. *ACM Trans. Comput. Educ.* 17, 4, 28.

[22] Anabela Gomes and Antonio José Mendes. 2007. Learning to program - difficulties and solutions *(International Conference on Engineering Education – ICEE 2007)*.

[23] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. 2013. Computational Thinking in Educational Activities: An Evaluation of the Educational Game Light-Bot *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 10–15.

[24] José Manuel Ruiz Gutiérrez. 2012. *S4A (Scratch) + Arduino: Utilización de S4A (Scratch) más la tarjeta Arduino en un ambiente de programación gráfica orientado a la educación*. Retrieved October 31, 2021 from http://math.tntech.edu/rafal/cliff11/index.html

[25] Keisuke Hattori and Tatsunori Hirai. 2019. An intuitive and educational programming tool with tangible blocks and AR. In *SIGGRAPH '19: ACM SIGGRAPH 2019 Posters*.

[26] Michael S. Horn and Robert J. K. Jacob. 2007. Designing tangible programming languages for classroom use. In *Proceedings of 1st international conference on Tangible and embedded interaction*.

[27] Qiao Jin, Danli Wang, Xiaozhou Deng, Nan Zheng, and Steve Chiu. 2018. ARmaze: a tangible programming tool for children based on AR technology. In *IDC '18: Proceedings of the 17th ACM Conference on Interaction Design and Children*.

[28] Sonya E. Keene. 1989. *Object-Oriented Programming in Common Lisp*. Addison-Wesley. http://cl-cookbook.sourceforge.net/clos-tutorial/

[29] Cátia Mesquita Brasil Khouri, Gidevaldo Novais dos Santos, and Maria Silva Santos Barbosa. 2020. Mapeamento Sistemático em Metodologias de Ensino-aprendizagem de Programação. *Revista de Ciência da Computação* 2, 1, 13–27.

[30] Barbara Kitchenham. 2004. *Procedures for Performing Systematic Reviews*. Joint Technical Report TR/SE-0401. Keele University, Keele.

[31] Wanda M. Kunkle and Robert B. Allen. 2016. The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts. *ACM Transactions on Computing Education* 16, 3(1–26). Issue 1.

[32] LEGO. 2016. *LEGO® Education WeDo 2.0 Computational Thinking Teacher's Guide*. Retrieved October 31, 2021 from https://education.lego.com/v3/assets/blt293eea581807678a/blt23ebdd9f2c7e9203/5f88042a6c54ba0f72c2085a/computationalthinkingteacherguide-en-us-v1.pdf

[33] LEGO. 2017. *LEGO® MINDSTORMS® Education EV3 Coding Activities Second Edition*. Retrieved October 31, 2021 from https://le-www-live-s.legocdn.com/downloads/LME-EV3/LME-EV3_Coding-activities_2.0_en-US.pdf

[34] Jiangjiang Liu, Cheng-Hsien Lin, Joshua Wilson, David Hemmenway, Ethan Hasson, Zebulun Barnett, and Yingbo Xu. 2014. Making Games a "Snap"with Stencyl: A Summer Computing Workshop for K-12 Teachers *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA.

[35] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. [n.d.]. Introductory Programming: A Systematic Literature Review.

[36] Alexandros Merkouris, Konstantinos Chorianopoulos, and Achilles Kameas. 2017. Teaching Programming in Secondary Education Through Embodied Computing Platforms: Robotics and Wearables. *ACM Trans. Comput. Educ.* 17, 2, 22.

[37] Amon Millner and Edward Baafi. 2011. Modkit: Blending and extending approachable platforms for creating computer programs and interactive objects. 250–253.

[38] Manasses Neto, Camila Santos, Edmar Souza, and Marcos Fonseca. 2018. Robótica educacional uma ferramenta para ensino de lógica de programação no ensino fundamental. In *Anais do XXIV Workshop de Informática na Escola* (Fortaleza, CE). SBC, Porto Alegre, RS, Brasil, 315–322.

[39] NI. [n.d.]. *What is LabVIEW?* Retrieved October 31, 2021 from https://www.ni.com/pt-br/shop/labview.html

[40] Daltro José Nunes. 2019. *Educação Superior em Computação – Estatísticas 2019*. Technical Report. Porto Alegre, RS. 67 pages.

[41] Martin Odersky, Lex Spoon, and Bill Venners. 2008. *Programming in Scala* (4th ed.). Artima. 776 pages.

[42] Erik Pasternak, Rachel Fenichel, and Andrew N. Marshall. 2017. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*.

[43] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A Survey of Literature on the Teaching of Introductory Programming. *ACM SIGCSE Bulletin* 39, 204–223. Issue 4.

[44] Diego Pereira, Rodrigo Seabra, and Adler Diniz de Souza. 2021. Ferramentas de Apoio ao Ensino Introdutório de Programação: um Mapeamento Sistemático. *RENOTE* 18, 491–500.

[45] Ana M Pinto-Llorente, Sonia Casillas Martín, Marcos Cabezas González, and Francisco José García-Peñalvo. 2016. Developing computational thinking via the visual programming tool: lego education WeDo. In *TEEM '16: Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*.

[46] Roger S. Pressman. 2016. *Engenharia de Software: Uma abordagem profissional* (8 ed.). Bookman, Porto Alegre. 968 pages.

[47] Thomas W. Price, Neil C.C. Brown, Dragan Lipovac, Tiffany Barnes, and Michael Kölling. 2016. Evaluation of a Frame-Based Programming Editor. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 33–42.

[48] Rubens Queiroz and Fábio Sampaio. 2016. DuinoBlocks for Kids: um ambiente de programação em blocos para o ensino de conceitos básicos de programação a crianças do Ensino Fundamental I por meio da Robótica Educacional. In *Anais do XXIV Workshop sobre Educação em Computação* (Porto Alegre). SBC, Porto Alegre, RS, Brasil, 91–100.

[49] Mitchel Resnick, John Harold Maloney, Andrés Monroy-Hernández, Natale Rusk, Evelyn Eastmond, Karen A Brennan, Amon Millner, Eric Rosenbaum, Jay Saul Silver, Brian S Silverman, and Yasmin Bettina Kafai. 2009. Scratch: programming for all. In *Communications of the ACM*.

[50] Bernat Romagosa i Carrasquer. 2019. *The Snap! Programming System*. Springer International Publishing, Cham, 1–10.

[51] Peter Van Roy, Joe Armstrong, Matthew Flatt, and Boris Magnusson. 2003. The Role of Language Paradigms in Teaching Programming. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*. Reno, Nevada, 269–270.

[52] S. P. Sarkar, B. Sarker, and S. K. A. Hossain. 2016. Cross platform interactive programming learning environment for kids with edutainment and gamification. In *2016 19th International Conference on Computer and Information Technology (ICCIT)*.

[53] Robert W. Sebesta. 2018. *Conceitos de linguagens de programação* (11 ed.). Bookman, Porto Alegre. 765 pages.

[54] Plácido Segundo, Mysael Carvalho, Osias Santos, Bruno Serejo, Joao Diniz, and Neilson Ribeiro. 2019. PENSAMENTO COMPUTACIONAL: Uma estratégia de ensino e promoção da cidadania na educação básica indígena utilizando robótica livre e lógica de programação Scratch. In *Anais do XXV Workshop de Informática na Escola* (Brasília). SBC, Porto Alegre, RS, Brasil, 1374–1378.

[55] Rafael J. Segura, Francisco J. del Pino, Carlos J. Ogáyar, and Antonio J. Rueda. 2020. VR-OCKS: A virtual reality game for learning the basic concepts of programming. *Computer Applications in Engineering Education* 28, 1, 31–41.

[56] Mazyar Seraj, Serge Autexier, and Jan Janssen. 2018. BEESM, a block-based educational programming tool for end users. In *NordiCHI '18: Proceedings of the 10th Nordic Conference on Human-Computer Interaction*.

[57] Ravi Sethi. 1996. *Programming languages: concepts and constructs* (2 ed.). Addison-Wesley Publishing, Reading, MA. 640 pages.

[58] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. 2009. Analysis of Research into the Teaching and Learning of Programming. In *ICER '09: Proceedings of the fifth international workshop on Computing education research workshop*.

[59] Marina Silva Silva. 2021. Proposta de uma linguagem composicional visual para ensino de programação. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação), Universidade Federal do Pampa.

[60] Marina Silva Silva and Ana Paula Lüdtke Ferreira. 2021. Sintaxe baseada em gramáticas de grafos para uma linguagem funcional visual voltada ao aprendizado de programação. In *Anais do VI Workshop-Escola de Informática Teórica*. 120–127. https://eventos.unipampa.edu.br/weit2021/anais/

[61] Elliot Soloway. 1993. Should we Teach Students to Program? *Commun. ACM* 36, 10.

[62] Lucas Sousa, Eder Farias, and Windson Carvalho. 2020. Programação em Blocos Aplicada no Ensino do Pensamento Computacional: Um Mapeamento Sistemático. In *Anais do XXXI Simpósio Brasileiro de Informática na Educação* (Online). SBC, Porto Alegre, RS, Brasil, 1513–1522.

[63] Daniella Souza, Maria Goulart, Graziela Guarda, and Ione Goulart. 2018. Lightbot Logicamente: um game lúdico amparado pelo Pensamento Computacional e a Matemática. In *Anais do XXIV Workshop de Informática na Escola* (Fortaleza, CE). SBC, Porto Alegre, RS, Brasil, 61–69.

[64] J. Su and T. Lin. 2018. Building a Simulated Blockly-Arduino-Based Programming Learning Tool: A Preliminary Study. In *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*.

[65] Luciano Teran, Fabiola Araújo, and Yomara Pires. 2019. ELIS: Uma ferramenta inclusiva para o ensino de lógica de programação aos surdos. In *Anais do XXV Workshop de Informática na Escola* (Brasília). SBC, Porto Alegre, RS, Brasil, 1024–1033.

[66] Simon Thompson. 1999. *Haskell: The Craft of Functional Programming* (2 ed.). Addison-Wesley. 507 pages. https://www.haskell.org/

[67] Bruno Toledo, Davi Silva, Karina Lemos, and Marcos Toledo. 2020. *DESENVOLVIMENTO DE UM JOGO EDUCACIONAL PARA O ENSINO DE PROGRAMAÇÃO BÁSICA*. 112–125.

[68] David S. Touretzky. 1990. *COMMON LISP: A Gentle Introduction to Symbolic Computation*. The Benjamin/Cummings Publishing Company, Inc., Redwood City. 587 pages. http://www.inf.ufsc.br/~aldo.vw/func/touretzky/touretzky.pdf

[69] Sergio Trilles and Carlos Granell. 2018. El proyecto SUCRE4Kids: una iniciativa de hardware y software libre para la introducción a la programación. *Novática* 240, 2444-6629.

[70] Jeffrey D. Ullman. 1998. *Elements of ML Programming*. Prentice Hall, Englewood Cliffs, New Jersey. 383 pages.

[71] Adilson Vahldick, António Mendes, Maria Marcelino, and Paulo Farah. 2016. Pensamento Computacional Praticado com um Jogo Casual Sério no Ensino Superior. In *Anais do XXIV Workshop sobre Educação em Computação* (Porto Alegre). SBC, Porto Alegre, RS, Brasil, 308–317.

[72] Maja Videnovik, Elena Vlahu-Gjorgievska, and Vladimir Trajkovik. [n.d.]. To code or not to code: Introducing coding in primary schools. *Computer Applications in Engineering Education* n/a, n/a.

[73] Y. Wang, Y. Zhang, A. Mao, J. Wang, and N. Li. 2020. The Research of Programming Teaching in Primary School on the Cultivation of Computational Thinking*. In *2020 Ninth International Conference of Educational Innovation through Technology (EITT)*.

[74] Takeshi Watanabe, Yuriko Nakayama, Yasunori Harada, and Yasushi Kuno. 2020. Analyzing Viscuit Programs Crafted by Kindergarten Children *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 238–247.

[75] Robert Stuart Weiss. 1995. *Learning from Strangers: The Art and Method of Qualitative Interview Studies*. Free Press. 256 pages.

[76] Jeannette M. Wing. 2008. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society*.

[77] Jeannette M. Wing. 2014. Computational thinking benefits society.

[78] Eben B. Witherspoon, Christian D. Schunn, Ross M. Higashi, and Robin Shoop. 2018. Attending to structural programming features predicts differences in learning and motivation. *Journal of Computer Assisted Learning* 34, 2, 115–128. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.12219

[79] David Wolber. 2011. App Inventor and Real-World Motivation. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 601–606.

[80] Ebru Yilmaz Ince and Mustafa Koc. 2021. The consequences of robotics programming education on computational thinking skills: An intervention of the Young Engineer's Workshop (YEW). *Computer Applications in Engineering Education* 29, 1, 191–208. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cae.22321

[81] Weixin Zhang, Yaozhu Sun, and Bruno C. D. S. Oliveira. 2021. Compositional Programming. *ACM Trans. Program. Lang. Syst.* 43, 3, Article 9, 61 pages.