Análise de Code Smells em Linguagens Visuais de Programação no Contexto da Educação: Um Mapeamento Sistemático da Literatura

Gerson Fernandes Galante [1], **Jean C. R. Hauck**[1]

[1]Programa de Pós-Graduação em Ciência da Computação - INE/CTC/UFSC
Térreo - Sala 109 - Trindade - Florianópolis, SC - Brasil

gersongalante19@gmail.com, jean.hauck@ufsc.br

Resumo. *As Linguagens Visuais de Programação (VPLs) têm sido utilizadas no ensino do pensamento computacional. No entanto, o aprendizado de programação apresenta diversas dificuldades, mesmo quando se utilizam VPLs. Alunos iniciantes, muitas vezes não utilizam boas práticas de codificação, gerando possíveis problemas de design ou implementação chamados de Code Smells. Assim, este trabalho realiza um Mapeamento Sistemático de Literatura buscando identificar como Code Smells têm sido analisados em VPLs no contexto educacional. São selecionados 14 estudos primários que atendem aos critérios de inclusão e exclusão. Os resultados indicam que a maioria dos estudos é aplicada com estudantes da educação básica, utilizando Scratch ou App Inventor. A descoberta de Code Smells é realizada por meio de análise estática de código, os problemas encontrados são tratados por meio de refatoração de código, resultando em melhor compreensão dos alunos, formação de melhores hábitos de programação e identificação de dificuldades de aprendizagem.*
Palavras-chave*: Code Smell, Mapeamento Sistemático de Literatura, Linguagem de Programação Visual.*

Abstract. *Visual Programming Languages (VPLs) have been used to teach computational thinking. However, learning to program presents several difficulties, even when VPLs are used. Beginning students often do not use good coding practices, generating possible design or implementation problems called Code Smells. Thus, this work performs a Systematic Literature Mapping seeking to identify how Code Smells have been analyzed in VPLs in the educational context. Fourteen primary studies that meet the inclusion and exclusion criteria are selected. The results indicate that most studies are applied to basic education students, using Scratch or App Inventor. The discovery of Code Smells is performed through static code analysis, the problems found are addressed through code refactoring, resulting in better student understanding, formation of better programming habits and identification of learning difficulties.*
Keywords*: Code Smell, Systematic Literature Mapping, Visual Programming Language.*

1. Introdução

O pensamento computacional, como processo de formulação de problemas e expressão da sua solução, tem sido considerado essencial para todos, não apenas para cientistas

da computação [Espinal et al. 2024] [Wing 2021]. Nesse sentido, o pensamento computacional vem sendo considerado como parte do conjunto de habilidades básicas para o desenvolvimento analítico e criativo de crianças e jovens, ao lado da leitura, escrita e aritmética [Zanetti et al. 2016] [Wing 2021].

Assim, a introdução do pensamento computacional desde o Ensino Fundamental promove a inclusão digital, especialmente entre jovens em situação de vulnerabilidade social, ampliando oportunidades econômicas e sociais [Costa et al. 2016], proporcionando o desenvolvimento de habilidades do século XXI, como pensamento crítico, resolução de problemas e trabalho em equipe e tornando os estudantes mais preparados para desafios contemporâneos [Mioto et al. 2019].

Diversas iniciativas têm utilizado o ensino de programação por meio de Linguagens Visuais de Programação (VPLs) no desenvolvimento de habilidades do pensamento computacional com estudantes da Educação Básica [Yang et al. 2025][Wu et al. 2023][da Cruz Alves et al. 2019]. As VPLs oferecem vantagens para alunos iniciantes, pois reduzem a complexidade do aprendizado de sintaxes complexas das linguagens de programação baseadas em texto [Wu et al. 2023].

No entanto, o aprendizado de programação enfrenta diversas dificuldades, mesmo quando se utilizam VPLs [Yang et al. 2025]. Alunos iniciantes, muitas vezes, não utilizam boas práticas de codificação que promovam qualidade e manutenibilidade do código [Techapalokul and Tilevich 2015], gerando Code Smells, que consistem em características no código-fonte que indicam possíveis problemas de design ou implementação [Prestat et al. 2022].

Code Smells muitas vezes envolvem características estruturais de software que podem indicar um problema de código ou design e podem tornar o software difícil de evoluir e manter [Fontana et al. 2012]. Os Code Smells mais comuns são: Duplicated Code, Dead Code, Long Method, Spaghetti Code, Large Class, Blob, Feature Envy, God Class, Data Class, Long Parameter List, Shotgun Surgery, Switch Statement, Functional Decomposition, Swiss Army Knife, Refused Bequest, Lazy Class e Divergent Change [Fowler 2018]. Se não forem corrigidos, esses problemas podem afetar negativamente a manutenibilidade do código, a eficiência e a segurança do software a longo prazo. A análise de Code Smells no código-fonte produzido pelos alunos nas atividades de aprendizado de programação pode ajudar a reduzir esses problemas [Gutmann et al. 2023][Hermans and Aivaloglou 2016a].

Assim, surge a pergunta de pesquisa deste trabalho: ”Como são analisados os Code Smells de projetos desenvolvidos com VPLs no contexto educacional?”. Para responder a essa pergunta de pesquisa, é realizado um Mapeamento Sistemático da Literatura (MSL) sobre análise de Code Smells em VPLs, focado exclusivamente em contextos educacionais.

Seguindo um protocolo definido, são selecionados 14 estudos primários. A análise dos dados coletados indica que a maioria dos estudos é aplicada com estudantes da educação básica, que utilizam Scratch ou App Inventor. A descoberta de Code Smells é realizada majoritariamente por meio de análise estática de códigos, os problemas encontrados são tratados por meio de refatoração de código, resultando em melhor compreensão dos alunos, formação de melhores hábitos de programação e identificação de dificuldades

de aprendizagem. São duas as principais contribuições deste estudo secundário: (i) para pesquisadores é apresentado um mapeamento da literatura, representando o estado da arte acerca da análise de Code Smells em VPLs no contexto educacional; (ii) para educadores, este estudo fornece indicações de como Code Smells têm sido identificados, tratados e quais seus principais impactos para alunos iniciantes utilizando VPLs.

O restante da estrutura deste artigo está disposta como segue: a Seção 2 aborda os trabalhos relacionados, a Seção 3 apresenta o protocolo do MSL, a Seção 4 apresenta os resultados, seguida da Seção 5 de ameaças à validade, da Seção 6 de discussão e da Seção 7 onde estão as considerações finais do trabalho.

2. Trabalhos Relacionados

O tema de Code Smells no aprendizado de programação tem atraído a atenção da academia, incluindo algumas revisões de literatura.

O trabalho [Lacerda et al. 2020] apresenta um estudo terciário de literatura sobre Code Smells e refatoração, abordando pesquisas secundárias já publicadas e explorando ferramentas, técnicas de detecção e desafios. As conclusões do estudo apontam que há uma forte conexão entre Code Smells e refatoração, mas persistem lacunas sobre a automatização completa e o mapeamento de benefícios quantitativos, evidenciando oportunidades de pesquisa futura na área. Mesmo abrangente, esse estudo terciário não foca somente no contexto educacional.

Keuning et al. [Keuning et al. 2023] conduziram um estudo secundário focado na qualidade do código no contexto educacional em geral. O estudo identifica que o foco dos estudos primários tem sido o desenvolvimento de ferramentas para avaliação de Code Smells e sugestões de melhorias e refatorações. Os autores também observaram um número crescente de estudos direcionados a VPLs. No entanto, o estudo não busca entender como os Code Smells são analisados especificamente em VPLs.

Já o trabalho [Sonjaya and Munir 2025] analisa, por meio de uma revisão sistemática, como a adoção de VPLs baseadas em blocos (BBVP) facilita a compreensão de conceitos fundamentais e estimula a motivação de iniciantes. As conclusões apontam maior engajamento, fortalecimento do pensamento computacional e autoconfiança. Entretanto, este trabalho não aborda especificamente Code Smells.

Assim, apesar do crescente interesse na avaliação de Code Smells, no melhor do nosso conhecimento, não foi possível encontrar revisões de literatura que abordem especificamente a avaliação de Code Smells em VPLs no contexto de ensino de programação.

3. Metodologia

Este trabalho apresenta um Mapeamento Sistemático da Literatura (MSL) com o objetivo de investigar Code Smells em projetos desenvolvidos com VPLs no contexto do ensino de computação, seguindo a abordagem proposta por [Petersen et al. 2008], com aplicação de *backward* e *forward snowballing* [Badampudi et al. 2015]. Nesta seção, é apresentado o protocolo de pesquisa deste MSL.

Este MSL tem a seguinte questão principal de pesquisa: Como são analisados os Code Smells de projetos desenvolvidos com VPLs no contexto educacional?

A partir dessa pergunta de pesquisa, são derivadas as seguintes subquestões de pesquisa:

- SQ1. Como é o contexto no qual Code Smells são encontrados?
- SQ2. Quais são os Code Smells encontrados em linguagens VPLs?
- SQ3. Como os Code Smells são descobertos e analisados?
- SQ4. Como os Code Smells são tratados ou solucionados?
- SQ5. Quais os impactos da análise de Code Smells?
- SQ6. Quais são as linguagens VPLs nas quais os Code Smells são analisados?

Bases de dados: São realizadas pesquisas nas seguintes bases de dados: IEEEXplore, ACM Digital Library, Science Direct, SCOPUS e Google Acadêmico devido à sua relevância para as áreas de Engenharia de Software e Educação em Computação.

3.1. Critérios de Inclusão e Exclusão

A partir do objetivo e das perguntas de pesquisa, são definidos os seguintes critérios de inclusão e exclusão.

Critérios de Inclusão (CI): i) Apenas artigos que tratam de Code Smells; ii) Artigos no contexto educacional, especialmente focados no ensino de programação; iii) A linguagem de programação utilizada é visual (VPL); iv) Apenas estudos primários são incluídos; v) Estudos publicados a partir de janeiro de 1997 (início da criação de VPLs mais populares [Tempel 2013]).

Critérios de Exclusão (CE): i) Literatura Cinzenta: estudos que não tenham sido publicados em eventos ou periódicos científicos; ii) Estudos secundários; iii) Artigos Duplicados: artigos tratando do mesmo tema e com os mesmos autores; iv) Artigos em idiomas diferentes do inglês.

3.2. String de Busca

Com base na pergunta de pesquisa, são derivados os termos de busca e seus sinônimos. A string de busca foi refinada utilizando-se como referências alguns estudos primários previamente conhecidos como possivelmente relevantes para a pesquisa (tais como [Hermans et al. 2016] e [Vargas-Alba et al. 2019]). A Tabela 1 apresenta a string de busca genérica, posteriormente adaptada para a sintaxe de cada uma das bases de dados.

Tabela 1. String de Buscas Genérica

(”visual programming language”OR vpl OR ”block”OR ”scratch”OR ”app inventor”OR ”snap!”) AND (”code smell*”OR ”bad smell*”OR ”analysis”OR ”code quality”OR ”God Class”OR ”Large Class”OR ”Blob”OR ”Feature Envy”OR ”Long Method”OR ”Data class”OR ”Functional Decomposition”OR ”Spaghetti Code”OR ”Long Parameter List”OR ”Swiss Army Knife”OR ”Refused Bequest”OR ”Shotgun Surgery”OR ”Code clone”OR ”Duplicated code”OR ”Lazy Class”OR ”Divergent Change”OR ”Dead Code”OR ”Switch Statement”)

3.3. Execução da Busca e Seleção

A busca foi realizada de setembro a dezembro de 2024, seguindo o protocolo definido. A string de busca foi aplicada em cada uma das bases de dados, resultando em 1.372 resultados. A partir desses resultados obtidos, foram realizados dois ciclos de seleção. No

primeiro ciclo, os títulos e resumos foram lidos, aplicando-se os critérios de inclusão e exclusão. Como resultado, foram obtidos 62 resultados potencialmente relevantes. No segundo ciclo de seleção, os artigos potencialmente relevantes foram lidos na íntegra, novamente com a aplicação dos critérios de inclusão e exclusão. Foram, então, selecionados 12 estudos primários que passaram nos critérios de inclusão e exclusão, conforme apresenta a Tabela 2.

Tabela 2. Resultados da busca em bases de dados

Base de dados	Resultados Iniciais	Primeiro Ciclo	Segundo Ciclo
IEEEXplore	28	13	6
ACM Digital Library	251	21	2
Science Direct	75	0	0
SCOPUS	646	11	0
Google Acadêmico	372	17	4
Total	1.372	62	12

Nos ciclos de seleção, foram principalmente descartados artigos (i) cujo foco trata da avaliação da competência em pensamento computacional ou de outras formas de avaliação que não envolvem a análise de Code Smells em programação com VPLs ([Alves et al. 2020], [Martínez-Valdés and Martínez-Ijaji 2018], [Basu 2019]) e (ii) artigos que avaliam aspectos de usabilidade e design visual de aplicativos ([Solecki et al. 2020], [Rijo-Garcia et al. 2022]).

A partir dessa listagem de 12 estudos inicialmente selecionados, foi realizada manualmente a aplicação de *backward* e *forward snowballing* [Badampudi et al. 2015] (utilizando o Google Acadêmico). Seguindo os critérios de inclusão e exclusão, foram encontrados os seguintes estudos relevantes [Hermans and Aivaloglou 2016b] e [Techapalokul 2017], que ao final totalizaram 14 estudos. A Tabela 3 apresenta a listagem dos estudos primários selecionados. Os dados brutos extraídos dos estudos selecionados estão disponíveis em [Galante and Hauck 2025].

Tabela 3. Estudos Primários Selecionados

ID	Referência
S1	[Li et al. 2017]
S2	[Seo 2018]
S3	[Patton et al. 2019]
S4	[Hermans et al. 2016]
S5	[Vargas-Alba et al. 2019]
S6	[Techapalokul and Tilevich 2017b]
S7	[Techapalokul and Tilevich 2017a]
S8	[Kong et al. 2021]
S9	[Fronza et al. 2020]
S10	[Aivaloglou and Hermans 2016]
S11	[Rose et al. 2018]
S12	[Techapalokul and Tilevich 2015]
S12B	[Hermans and Aivaloglou 2016b]
S12F	[Techapalokul 2017]

4. Resultados

Esta seção apresenta a análise dos dados coletados. Os resultados são apresentados seguindo a ordem das subquestões de pesquisa definidas.

RQ1. Como é o contexto no qual Code Smells são encontrados?

Nesta pergunta de pesquisa, procura-se identificar nos estudos primários selecionados, qual o perfil dos alunos participantes dos estudos e em qual nível educacional são aplicados. Os participantes dos estudos são, em sua maioria, alunos da Educação Básica com idades entre 10 e 18 anos, que estão iniciando no aprendizado de programação. O tamanho da amostragem, no entanto, varia bastante entre os estudos.

Em [Kong et al. 2021] são analisados 27 projetos desenvolvidos por alunos da Educação Básica em tarefas específicas de aula. Já em [Li et al. 2017] o estudo envolve tanto estudantes quanto programadores casuais, analisando mais de 1,5 milhão de projetos de 10.000 usuários do App Inventor selecionados de forma aleatória. Em [Aivaloglou and Hermans 2016] foi conduzido um experimento com 61 alunos iniciantes, divididos em grupos com diferentes versões de um programa.

RQ2. Quais são os Code Smells encontrados em linguagens VPLs?

Esta pergunta de pesquisa objetiva identificar a frequência dos Code Smells identificados nos estudos primários. Como os estudos utilizam diferentes nomenclaturas para os Code Smells, foi utilizada a nomenclatura definida por [Fowler 2018] para categorizá-los. A Tabela 4 apresenta um compilado dos Code Smells e a frequência com que aparecem nos estudos selecionados.

Tabela 4. Code Smells Encontrados nos Estudos Selecionados

Code Smell	Qtd	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S12B	S12F
Duplicated Code	12	X		X	X	X	X	X		X	X	X	X	X	X
Dead Code	9	X			X	X	X	X	X		X	X			X
Long Method	8				X		X			X	X	X	X	X	X
Code Clone	4	X	X	X											X
Feature Envy	3				X		X								X
Large Class	1										X				
Spaghetti Code	1												X		
Long Param List	1				X										
Lazy Class	1				X										

É possível observar na Tabela 4 que os Code Smells mais recorrentes em VPLs incluem *Duplicated Code* (12 ocorrências), onde os alunos tendem a replicar blocos em vez de criar abstrações adequadas [Hermans and Aivaloglou 2016b]; seguido da existência de *Dead Code*, que consiste em blocos que nunca são executados (9 ocorrências) [Li et al. 2017] e a presença de *Long Method* – Scripts com muitos blocos (8 ocorrências) [Aivaloglou and Hermans 2016] [Fronza et al. 2020].

RQ3. Como os Code Smells são descobertos e analisados?

Nesta questão, busca-se identificar de que forma os Code Smells são encontrados e analisados nos códigos em VPLs desenvolvidos pelos alunos.

A detecção de Code Smells em VPLs baseadas em blocos é realizada predominantemente por meio da análise estática automatizada dos blocos de código. No entanto, a descoberta de Code Smells em ambientes de programação e ensino de computação varia em complexidade e abordagem conforme o tipo de ferramenta e o objetivo da análise.

Patton et al. [Patton et al. 2019] utilizam um método de análise estática automatizada para identificar alguns tipos de Code Smells no App Inventor. Seu algoritmo compara as árvores de sintaxe abstrata (ASTs) dos blocos de código para encontrar manipuladores de eventos duplicados e detectar oportunidades de abstração, como o uso de procedimentos genéricos. Para isso, os autores desenvolveram uma ferramenta em Python que converte o código em VPL do App Inventor em formato JSON (JAIL), facilitando a detecção de padrões indesejáveis no código [Patton et al. 2019]. Já Seo [Seo 2018] desenvolve um programa para varrer todos os manipuladores de eventos e encontrar corpos de blocos duplicados (*"Duplicated Code"*).

No caso do Code Smell *Long Method*, a detecção é realizada via métrica de contagem de blocos em um mesmo *handler*. Estudos como [Aivaloglou and Hermans 2016] definiram um limiar (ex: mais de 10 blocos) para classificar um script como "longo" e usaram scripts para contar blocos nos projetos VPLs.

Techapalokul et al. [Techapalokul and Tilevich 2017b] agrupam, categorizam e ordenam os Code Smells do mais ao menos prevalente nos projetos Scratch analisados. Além disso, os autores buscam compreender como a frequência dos problemas de qualidade recorrentes impacta no compartilhamento de código baseado em blocos do Scratch e afeta a probabilidade de um determinado código ser remixado e estendido.

De forma geral, a análise dos Code Smells é realizada de forma multifacetada: (i) quantitativa: para estabelecer quais Code Smells são mais comuns e em que proporção; (ii) comparativa: para correlacionar com a experiência ou avaliar diferenças entre grupos; (iii) pedagógica: para avaliar impactos em aprendizagem e dificuldades; e (iv) causal: para compreender os motivos e possíveis soluções. Essa variedade de análises fornece um panorama rico, pois os estudos buscam inferir o significado dos problemas identificados no contexto educacional e o que esses problemas revelam sobre o aluno ou a VPL/ferramenta/plataforma utilizada.

RQ4. Como os Code Smells são tratados ou solucionados?

Nesta pergunta, são buscadas informações nos estudos primários sobre como apoiar os alunos na solução dos Code Smells que são identificados no contexto educacional.

Na sua maioria, os Code Smells identificados são tratados principalmente por meio de refatoração de código, utilizando práticas como criação de blocos customizados (procedimentos), *loops* e remoção de código redundante, visando aprimorar a estrutura do código e evitar práticas prejudiciais [Vargas-Alba et al. 2019] [Techapalokul and Tilevich 2017a].

Do ponto de vista de ferramentas, uma linha de solução propõe incorporar suporte à refatoração automática nas próprias plataformas que utilizam VPLs. [Techapalokul and Tilevich 2017a] apresentam um protótipo de ambiente Scratch aprimorado, que oferece refatorações com um clique para eliminar Code Smells recorrentes.

Eles implementaram diferentes tipos de refatoração (por exemplo, extrair procedimentos para remover código duplicado (*"Duplicated Code"*), eliminar blocos mortos (*"Dead Code"*), etc.) encontrando alta aplicabilidade e melhorias de qualidade de código após as transformações.

RQ5. Quais os impactos da análise de Code Smells?

Os estudos primários selecionados indicam que a análise de Code Smells em VPLs em ambientes educacionais traz impactos importantes tanto para a aprendizagem dos alunos quanto para a evolução das ferramentas e estratégias de ensino. A seguir, destacam-se os principais efeitos e implicações observados.

Quanto ao **impacto no desempenho e compreensão dos alunos**: A presença de Code Smells pode prejudicar o aprendizado. O experimento controlado de [Hermans and Aivaloglou 2016b] demonstrou que estudantes iniciantes rendem pior quando trabalham com código que contém Smells. Especificamente, os alunos que trabalham em códigos que incluem Code Smells têm um desempenho significativamente pior nas tarefas propostas em comparação com aqueles que receberam um código equivalente sem esses problemas [Aivaloglou and Hermans 2016]. A presença conjuntos de blocos longos (*Long Method)* reduziu a capacidade dos alunos entenderem o programa como um todo, enquanto a duplicação de blocos (*Duplicated Code*) aumentou a dificuldade para modificar/estender os programas [Aivaloglou and Hermans 2016].

Quanto à **formação de melhores hábitos de programação**: Quando alunos tomam consciência de Code Smells e aprendem a corrigi-los, desenvolvem melhores práticas que serão úteis em aprendizagem avançada. Por exemplo, [Techapalokul and Tilevich 2017b] notaram que oferecer sugestões de melhoria de código aumentou a propensão de estudantes iniciantes considerarem a qualidade e aplicarem refatorações. Mesmo após curto período, os participantes passaram a enxergar alternativas para evitar repetição e valorizaram código mais claro.

Quanto à **Identificação de dificuldades de aprendizado ocultas**: A detecção de Code Smells pode revelar onde alunos estão tendo problemas conceituais. O trabalho de [Kong et al. 2021] mostra que analisar código morto ajudou a diagnosticar falhas de compreensão sobre laços e estruturas de dados – alunos deixavam blocos de lista desconectados porque não sabiam integrá-los corretamente. Sem essa análise, o professor poderia notar que o programa não funciona bem, mas não saber exatamente a causa. Assim, os professores podem focar em conceitos onde muitos Code Smells são detectados (ex.: "*vejo que vários de vocês duplicaram este trecho – vamos aprender uma forma mais elegante de fazer isso?*").

RQ6. Quais são as linguagens VPLs nas quais os Code Smells são analisados?

As linguagens/ambientes/plataformas VPLs mais predominantes nos estudos são apresentadas na Tabela 5.

Tabela 5. VPLs utilizadas nos Estudos Primários

VPL	Qtd	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S12B	S12F
Scratch	9					X	X	X		X	X	X	X	X	X
App Inventor	4	X	X	X					X						
LEGO Mindstorms	1				X										
Microsoft Kodu	1				X										
Snap!	1												X		

A Tabela 5 mostra que o Scratch é a linguagem/plataforma de programação visual (VPL) mais frequentemente estudada nos artigos, representando mais da metade (9 estudos - 56,25%) dos estudos primários selecionados, seguido pelo App Inventor, abordado em 4 estudos (28,57%). Em contraste, outras plataformas como LEGO Mindstorms EV3 e Microsoft Kodu aparecem com baixa frequência, indicando que, apesar de serem relevantes no contexto educacional, recebem menor atenção em pesquisas voltadas para a identificação e análise de Code Smells. A predominância do Scratch e App Inventor evidencia uma tendência das pesquisas em se concentrarem em plataformas amplamente adotadas em contextos educativos.

5. Ameaças à Validade

Nesta seção, as potenciais ameaças à validade deste MSL são discutidas. Como tipicamente ocorre em MSLs, a omissão de trabalhos relevantes é um risco significativo. Para minimizá-lo, foram adotadas estratégias como a seleção rigorosa de palavras-chave e a revisão cruzada de todas as etapas de seleção entre os autores. Além disso, a string de busca foi exaustivamente testada antes de sua versão final para aplicação, incluindo-se termos abrangentes e sinônimos.

O viés de publicação é uma ameaça, mas seu impacto nesta pesquisa é minimizado, pois o foco deste estudo secundário está na caracterização das abordagens de avaliação de Code Smells em projetos utilizando VPLs no contexto de ensino, e não na sua eficácia. Para mitigar possível viés, foram usadas múltiplas bases de dados e incluídos estudos com resultados diversos.

A subjetividade na seleção e extração de dados pode afetar a confiabilidade dos resultados. Para mitigar isso, foram definidos critérios precisos de inclusão e exclusão, e um protocolo que guiou a seleção dos artigos. A extração dos dados foi realizada sistematicamente pelo primeiro autor e revisada pelo segundo autor. Contudo, pode haver subjetividade devido a inferências necessárias e à qualidade das informações extraídas.

6. Discussão

Os 14 artigos selecionados apresentam estudos primários que envolvem, em sua maioria, alunos da Educação Básica iniciantes em programação. Esse público-alvo dos estudos primários pode ser justificado por este MSL focar em Code Smells em VPLs, que tipicamente têm sido utilizadas no ensino de programação para iniciantes [Yang et al. 2025][Wu et al. 2023].

Os Code Smells identificados com maior frequência em VPLs: duplicação de código (*Duplicated Code*), código morto (*Dead Code*) e blocos de código longos (*Long*

Method) corroboram os achados de outros estudos secundários na área da educação [Keuning et al. 2023] e também na indústria [Lacerda et al. 2020], que abrangem todos os tipos de linguagens textuais de programação. No entanto, é importante destacar que esses Code Smells têm sido considerados dentre os mais fáceis de serem identificados no código de forma automatizada e isso pode tender à descoberta mais frequente desses problemas [Lacerda et al. 2020].

A análise de Code Smells em VPLs no contexto da educação tem sido principalmente realizada por meio de análise estática de código de forma automatizada. Esta também tem sido a forma mais comum de analisar Code Smells em outros tipos de linguagens de programação textuais [Rasool and Arshad 2015]. No entanto, alguns estudos priorizam a análise manual ou a comparação com a opinião de especialistas [Keuning et al. 2023], combinando análise dinâmica e estática, pois nem todos os Code Smells podem ser identificados somente por meio de análise estática [Ligu et al. 2013].

O tratamento e busca de solução dos Code Smells em VPLs na educação, observado nos estudos selecionados, tem sido realizado por meio da aplicação de refatoração de código, tanto manual quanto automatizada. Essa tem sido a forma mais comum de tratamento de Code Smells em outros contextos de aplicação [Lacerda et al. 2020]. No entanto, a refatoração de código nem sempre é capaz de remover os Code Smells, especialmente aqueles não estruturais [Yoshida et al. 2016].

Os principais resultados observados da análise de Code Smells em VPLs na educação são: (i) melhoria na compreensão dos alunos, (ii) formação de melhores hábitos de programação e (iii) identificação de dificuldades de aprendizado ocultas. Esses resultados positivos observados complementam outros estudos que também identificaram como benefícios: melhorias na legibilidade de código, melhoria na compreensão de conceitos e aumento da criatividade [Keuning et al. 2023]. Esses impactos educacionais positivos podem prevenir que maus hábitos se consolidem, propiciando um feedback formativo [Thangaraj et al. 2023] e influenciando positivamente o programador iniciante.

7. Conclusão

Este artigo apresenta um Mapeamento Sistemático de Literatura sobre a análise de Code Smells em Linguagens Visuais de Programação (VPLs) aplicadas ao contexto educacional. São selecionados 14 estudos primários que abordam diretamente a avaliação desses problemas de qualidade de código em ambientes de ensino.

Os dados coletados mostram Code Smells recorrentes nos projetos desenvolvidos com VPLs, evidenciando a necessidade de intervenções para promover melhores práticas de programação. A análise dos dados coletados revelou que os Code Smells mais frequentes são duplicação de código, sequências longas de blocos, falta de modularização e falta de abstração. Também se pode notar que a utilização de análise estática de Code Smells em VPLs é eficaz para tornar o feedback compreensível para os alunos.

Esta pesquisa é um primeiro passo no desenvolvimento de uma ferramenta automatizada de avaliação de Code Smells em VPLs, que permitirá aos instrutores utilizarem de forma prática os resultados deste MSL para auxiliar seus alunos a evitarem Code Smells ou reduzirem seus impactos.

Referências

Aivaloglou, E. and Hermans, F. (2016). How kids code and how we know: An exploratory study on the scratch repository. In *Proceedings of the 2016 ACM conference on international computing education research*, pages 53–61.

Alves, N. D. C. et al. (2020). A large-scale evaluation of a rubric for the automatic assessment of algorithms and programming concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 556–562, Portland, OR, USA. ACM. Acesso em: 3 set. 2024.

Badampudi, D., Wohlin, C., and Petersen, K. (2015). Experiences from using snowballing and database searches in systematic literature studies. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, EASE '15, New York, NY, USA. Association for Computing Machinery.

Basu, S. (2019). Using rubrics integrating design and coding to assess middle school students' open-ended block-based programming projects. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1211–1217, Minneapolis, MN, USA. ACM. Acesso em: 3 set. 2024.

Costa, T. et al. (2016). A importância da computação para alunos do ensino fundamental: Ações, possibilidades e benefícios. In *Anais do Workshop de Informática na Escola*, pages 593–601. Acesso em: 7 nov. 2024.

da Cruz Alves, N., von Wangenheim, C. G., and Hauck, J. C. (2019). Approaches to assess computational thinking competences based on code analysis in k-12 education: A systematic mapping study. *Informatics in Education*, 18(1):17–39.

Espinal, A., Vieira, C., and Magana, A. J. (2024). Professional development in computational thinking: A systematic literature review. *ACM Trans. Comput. Educ.*, 24(2).

Fontana, F. A., Braione, P., and Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *J. Object Technol.*, 11(2):5–1.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Boston, 2nd edition.

Fronza, I., Corral, L., and Pahl, C. (2020). An approach to evaluate the complexity of block-based software product. inform educ 19 (1): 15–32.

Galante, G. and Hauck, J. (2025). Análise de code smells em linguagens visuais de programação no contexto da educação: Um mapeamento sistemático da literatura (msl). Mendeley Data, V1. DOI: `https://doi.org/10.17632/bmpfnwxzxd.1`.

Gutmann, V., Starke, E., and Michaeli, T. (2023). Investigating code smells in k-12 students' programming projects: Impact on comprehensibility and modifiability. In *International Conference on Informatics in Secondary Schools*.

Hermans, F. and Aivaloglou, E. (2016a). Do code smells hamper novice programming? a controlled experiment on scratch programs. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10.

Hermans, F. and Aivaloglou, E. (2016b). Do code smells hamper novice programming? a controlled experiment on scratch programs. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE.

Hermans, F., Stolee, K. T., and Hoepelman, D. (2016). Smells in block-based programming languages. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 68–72. IEEE.

Keuning, H., Jeuring, J., and Heeren, B. (2023). A systematic mapping study of code quality in education. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, page 5–11, New York, NY, USA. Association for Computing Machinery.

Kong, S. C., POON, C. W., and Bowen, L. (2021). Analysing reachable and unreachable codes in app inventor programs for supporting the assessment of computational thinking concepts. In *International Conference on Computers in Education*.

Lacerda, G., Petrillo, F., Pimenta, M., and Guéhéneuc, Y. G. (2020). Code smells and refactoring: A tertiary systematic review of challenges and observations. *Journal of Systems and Software*, 167:110610.

Li, I., Turbak, F., and Mustafaraj, E. (2017). Calls of the wild: Exploring procedural abstraction in app inventor. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 79–86. IEEE. Acesso em: 4 set. 2024.

Ligu, E., Chatzigeorgiou, A., Chaikalis, T., and Ygeionomakis, N. (2013). Identification of refused bequest code smells. In *2013 IEEE International Conference on Software Maintenance*, pages 392–395.

Martínez-Valdés, J. A. and Martínez-Ijaji, N. A. (2018). An experience with the app inventor in cs0 for the development of the stem didactics. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 51–56, Salamanca, Spain. ACM. Acesso em: 3 set. 2024.

Mioto, F. et al. (2019). Bases21 - um modelo para a autoavaliação de habilidades do século xxi no contexto do ensino de computação na educação básica. *Revista Brasileira de Informática na Educação*, 27(01):26. [s. l.].

Patton, E. W., Seo, A., and Turbak, F. (2019). Enhancing abstraction in app inventor with generic event handlers. In *2019 IEEE Blocks and Beyond Workshop (B&B)*, pages 63–71. IEEE.

Petersen, K. et al. (2008). Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. BCS Learning & Development. Acesso em: 14 nov. 2024.

Prestat, D., Moha, N., and Villemaire, R. (2022). An empirical study of android behavioural code smells detection. *Empirical Software Engineering*, 27(7):179. [s. l.].

Rasool, G. and Arshad, Z. (2015). A review of code smell mining techniques. *Journal of Software: Evolution and Process*, 27:867 – 895.

Rijo-Garcia, S., Segredo, E., and Leon, C. (2022). Computational thinking and user interfaces: A systematic review. *IEEE Transactions on Education*, 65(4):647–656. [s. l.].

Rose, S., Habgood, J., and Jay, T. (2018). Pirate plunder: Game-based computational thinking using scratch blocks. In *Academic Conferences and Publishing International Limited*. Academic Conferences and Publishing International Limited.

Seo, A. (2018). Abstractionless programming in app inventor. In *BLOCKS+ 2018 ACM SPLASH Workshop*. Acesso em: 3 set. 2024.

Solecki, I. et al. (2020). Automated assessment of the visual design of android apps developed with app inventor. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 51–57, Portland, OR, USA. ACM. Acesso em: 8 nov. 2024.

Sonjaya, R. P. and Munir (2025). Examining the impact of block-based visual programming in programming education: A systematic review. *Indonesian Journal of Teaching in Science*, 5(1):11–20.

Techapalokul, P. (2017). Sniffing through millions of blocks for bad smells. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 781–782.

Techapalokul, P. and Tilevich, E. (2015). Programming environments for blocks need first-class software refactoring support: A position paper. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 109–111, Atlanta, GA, USA. IEEE. Acesso em: 3 set. 2024.

Techapalokul, P. and Tilevich, E. (2017a). Quality hound—an online code smell analyzer for scratch programs. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 337–338. IEEE.

Techapalokul, P. and Tilevich, E. (2017b). Understanding recurring quality problems and their impact on code sharing in block-based software. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 43–51. IEEE.

Tempel, M. (2013). Blocks programming. *CsTA Voice*, 9(1):3–4.

Thangaraj, J., Ward, M., and O'Riordan, F. (2023). A Systematic Review of Formative Assessment to Support Students Learning Computer Programming. In Peixoto de Queirós, R. A. and Teixeira Pinto, M. P., editors, *4th International Computer Programming Education Conference (ICPEC 2023)*, volume 112 of *Open Access Series in Informatics (OASIcs)*, pages 7:1–7:13, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Vargas-Alba, Á., Troiano, G. M., Chen, Q., Harteveld, C., and Robles, G. (2019). Bad smells in scratch projects: A preliminary analysis. In *TACKLE@ EC-TEL*.

Wing, J. M. (2021). Pensamento computacional. In [s. n.], editor, *Educação e Matemática*, number 162 in Série A, pages 2–4. [s. l.].

Wu, T.-T., Lin, C.-J., Wang, S.-C., and Huang, Y.-M. (2023). Tracking visual programming language-based learning progress for computational thinking education. *Sustainability*, 15(3).

Yang, Z., Blake-West, J., Yang, D., and Bers, M. (2025). The impact of a block-based visual programming curriculum: Untangling coding skills and computational thinking. *Learning and Instruction*, 95:102041.

Yoshida, N., Saika, T., Choi, E., Ouni, A., and Inoue, K. (2016). Revisiting the relationship between code smells and refactoring. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–4.

Zanetti, H., Borges, M., and Ricart, I. (2016). Pensamento computacional no ensino de programação: Uma revisão sistemática da literatura brasileira. In *Anais do XXVII Simpósio Brasileiro de Informática na Educação*, page 21, Uberlândia, Minas Gerais, Brasil. [s. n.]. Acesso em: 15 abr. 2023.