Feedback Automatizado na Educação em Programação: Uma Revisão Estruturada da Literatura sobre Ferramentas e Lacunas no Suporte à Qualidade de Código

Lucas Moreira Rios†
Programa de Pós-graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
rios.l@pucpr.edu.br

Andreia Malucelli
Programa de Pós-graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
andreia.malucelli@pucpr.br

Sheila Reinehr
Programa de Pós-graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
sheila.reinehr@pucpr.br

Resumo

A educação em programação enfrenta desafios recorrentes relacionados à escalabilidade da avaliação e à eficácia do *feedback* fornecido aos alunos. Em turmas grandes e heterogêneas, a revisão manual de código torna-se limitada, comprometendo tanto o tempo de resposta quanto a profundidade do *feedback*, que são fatores críticos para o aprendizado. Nesse contexto, Ferramentas de *Feedback* Automatizado (AFTs) foram propostas para mitigar essas limitações, variando de técnicas tradicionais de análise estática até abordagens recentes baseadas em Inteligência Artificial (AI) *e Large Language Models* (LLMs). Este estudo apresenta um mapeamento estruturado da literatura com o objetivo de analisar o estado atual das AFTs, com foco especial no suporte à qualidade de código no ensino de programação. A partir de um conjunto inicial de 8.826 estudos, 17 ferramentas publicadas entre 2020 e 2025 foram selecionadas e analisadas. Os resultados indicam uma predominância de abordagens baseadas em AI e LLMs (41,17% das soluções) e uma alta disponibilidade de *feedback* imediato (88,2%). No entanto, observa-se uma lacuna significativa no suporte à qualidade estrutural do código. Enquanto o estilo de codificação (*coding style*) e heurísticas locais são amplamente abordados, o suporte ao *design* de *software* e à detecção de *code smells* permanece limitado e fragmentado. Como contribuições, este estudo analisa as abordagens existentes, examina criticamente o uso de LLMs em ambientes educacionais e destaca o desalinhamento entre o ensino de atributos de qualidade e o suporte fornecido pelas ferramentas atuais. Essas descobertas apontam para oportunidades de desenvolvimento de soluções que permitam um *feedback* mais profundo e orientado à arquitetura, contribuindo para uma educação mais robusta em Engenharia de *Software*.

Palavras-chave

Ensino de Engenharia de *Software*, Ferramenta de *Feedback* Automatizado, Qualidade de *Software*, *Code Smells*.

1 Introdução

A programação é uma atividade central na Engenharia de *Software*, sendo responsável não apenas pela materialização de requisitos funcionais e não funcionais, mas também por refletir diretamente nos atributos de qualidade dos sistemas produzidos [1]. A forma como o código é concebido, estruturado e mantido influencia diretamente aspectos como manutenibilidade, legibilidade, desempenho e robustez [2][3]. Assim, formar profissionais capazes de escrever código com boas características de qualidade é um objetivo essencial e um desafio constante nos cursos de graduação da área, dada a complexidade de transpor conceitos teóricos de boas práticas para a escrita de código real [4][5].

Ensinar programação a estudantes iniciantes constitui um desafio recorrente. Turmas numerosas, heterogeneidade de perfis e a consequente sobrecarga docente dificultam a revisão manual de código, comprometendo a tempestividade e a profundidade do *feedback*. Em muitos casos, a devolutiva ocorre dias após o prazo da tarefa, o que reduz drasticamente seu impacto pedagógico, uma vez que os estudantes já se encontram envolvidos em novas demandas cognitivas [6].

Diante desse cenário, diversas ferramentas de *feedback* automatizado (*Automatic Feedback Tools* - AFT*s*) têm surgido para mitigar essa lacuna. Contudo, essas soluções apresentam estratégias tecnológicas heterogêneas, variando desde análise estática tradicional até o uso recente de *Large Language Models* (LLMs).

Embora revisões anteriores tenham investigado o uso de *feedback* automatizado no ensino de programação, observa-se que o suporte automatizado à qualidade estrutural do código ainda recebe atenção limitada em contextos educacionais. Em particular, aspectos relacionados à modularização, manutenibilidade e detecção de *code smells* permanecem menos explorados quando comparados ao *feedback* voltado à corretude funcional, sintaxe ou estilo de código. *Code smells* correspondem a indícios de problemas de *design* ou implementação que, embora não representem necessariamente falhas funcionais, dificultam a manutenção e evolução do *software* [7].

Essa lacuna motiva a necessidade de investigar em que medida as AFTs atuais conseguem oferecer suporte pedagógico a aspectos estruturais da qualidade de *software*. Este trabalho apresenta um

mapeamento da literatura, conduzido de forma estruturada, com o objetivo de identificar o cenário atual, categorizar as tecnologias disponíveis e evidenciar as lacunas existentes no suporte automatizado ao ensino de programação, considerando aspectos de qualidade de *software*. Busca-se compreender em que medida as ferramentas atuais conseguem evoluir do *feedback* focado apenas em sintaxe e estilo para análises relacionadas à qualidade estrutural do código. Neste estudo, considera-se qualidade estrutural como aspectos relacionados à modularização, manutenibilidade e presença de *code smells*.

As principais contribuições deste trabalho incluem:

(i) Caracterização das AFTs quanto ao tipo de *feedback*, integração ao fluxo de desenvolvimento e suporte à qualidade estrutural;

(ii) Identificação de lacunas no cenário atual, evidenciando limitações recorrentes no suporte automatizado a aspectos estruturais do código, como a detecção de *code smells*;

(iii) Discussão crítica sobre o uso de LLMs no ensino de programação, explorando os desafios pedagógicos associados ao equilíbrio entre assistência automatizada e engajamento cognitivo do estudante.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica sobre *feedback* formativo e qualidade de *software*; a Seção 3 detalha o método utilizado, incluindo as questões de pesquisa, termos de busca e critérios de seleção; a Seção 4 apresenta os resultados e a discussão das lacunas identificadas; por fim, a Seção 5 apresenta as conclusões e sugestões para trabalhos futuros.

2 Fundamentação

2.1 *Feedback* Formativo e a Importância da Tempestividade

O *feedback* é reconhecido como um dos pilares fundamentais no processo educativo, atuando não apenas como métrica de desempenho, mas também como mecanismo de orientação para o alcance dos resultados de aprendizagem esperados [8]. No ensino de programação, a eficácia desse suporte está intrinsecamente ligada à sua tempestividade, ou seja, o momento oportuno em que ele ocorre. O fornecimento de *feedback* imediato desempenha um papel fundamental para reduzir o tempo de correção de erros e evitar a consolidação de modelos mentais inadequados [9][10].

Como destacam Liu et al. [11], a revisão manual de código gerado pelo estudante frequentemente ocorre dias após a entrega, momento em que ele já deslocou seu foco cognitivo para a tarefa seguinte, o que reduz drasticamente o impacto pedagógico da intervenção. Segundo [12], estudantes que recebem orientações em tempo real tendem a interagir significativamente mais com o *feedback* do que aqueles que o recebem dias depois.

Nesse sentido, Klinik, Koopman e van der Wal [6] argumentam que o efeito ideal sobre a aprendizagem ocorre quando o suporte é oferecido no exato momento em que ele mais importa para o estudante. O *feedback* automatizado, portanto, surge para evitar a consolidação de maus hábitos de desenvolvimento que poderiam persistir caso o erro não fosse apontado durante a execução da tarefa.

2.2 Qualidade de *Software* e o Desafio da Adaptação Pedagógica

A preocupação com a qualidade do código não deve ser encarada apenas como uma exigência da indústria, mas como um componente crítico de todo o ciclo de vida do desenvolvimento de *software*. A negligência com atributos internos, como legibilidade e manutenibilidade, resulta em sistemas vulneráveis a *bugs*, falhas de segurança e ciclos de manutenção onerosos [13].

No contexto educacional, a qualidade estrutural do código desempenha um papel relevante não apenas para a manutenção do *software*, mas também para a formação de hábitos de desenvolvimento nos estudantes. A ausência de *feedback* sobre problemas de modularização, acoplamento excessivo ou duplicação de código pode levar à consolidação de práticas inadequadas que persistem ao longo da formação acadêmica. Indicadores como os *code smells* podem atuar como mecanismos pedagógicos para apoiar a conscientização sobre decisões de *design* e organização do código. Dessa forma, os estudantes são incentivados a refletir não apenas sobre a corretude funcional de suas soluções, mas também sobre aspectos relacionados à manutenibilidade e evolução do *software*.

Apesar da existência de ferramentas consolidadas para análise automática de qualidade no mercado profissional, como os *linters* (ferramentas voltadas à verificação automática de convenções e padrões de código), observa-se que a maioria dessas soluções não foi concebida sob uma perspectiva pedagógica [14]. Enquanto ferramentas utilizadas na indústria focam na correção direta do código, visando produtividade, no contexto educacional o objetivo central deve ser o desenvolvimento da capacidade do estudante de compreender e refletir sobre aspectos estruturais do código. Essa disparidade justifica a necessidade de investigar *AFTs* que consigam traduzir métricas técnicas em orientações formativas acessíveis a estudantes iniciantes.

3 Método de Pesquisa

Este estudo apresenta um mapeamento da literatura conduzido de forma estruturada, inspirado nas diretrizes propostas por Kitchenham e Charters [15]. O processo foi estruturado para ampliar a transparência na identificação, seleção e análise das ferramentas investigadas, buscando reduzir vieses e aumentar a rastreabilidade do estudo.

3.1 Questões de Pesquisa (RQs)

A pesquisa é orientada pelas seguintes questões:

RQ1: Quais tecnologias de análise e geração de *feedback* são utilizadas nas ferramentas de *feedback* automatizado atuais?

RQ2: Como as ferramentas de *feedback* automatizado abordam atributos de qualidade e boas práticas de Engenharia de *Software*?

RQ3: Como as ferramentas de *feedback* automatizado se integram ao fluxo de desenvolvimento e interação do estudante?

3.2 Estratégia de Busca e Critérios de Inclusão/Exclusão

Para a identificação dos estudos primários, foram conduzidas buscas estruturadas nas bases ACM Digital Library, Scopus, ScienceDirect e IEEE Xplore. A *string* foi estruturada em três

dimensões: (i) *feedback* automatizado, (ii) contexto de desenvolvimento de *software* e (iii) análise de código. Os termos relacionados à educação foram eliminados para ampliar a possibilidade de trazer ferramentas mais abrangentes de *feedback* e potencialmente aplicáveis ao contexto educacional. As ferramentas puramente industriais foram posteriormente descartadas pelo Critério de Exclusão 2 (CE2), descrito adiante. Essa estratégia permitiu ampliar a recuperação de ferramentas com potencial aplicação educacional. Posteriormente, ferramentas exclusivamente industriais foram removidas por meio dos critérios de exclusão. A adequação da *string* foi avaliada iterativamente por meio de buscas piloto e inspeção preliminar dos resultados recuperados, entre os autores do estudo.

("Feedback" OR "Recommendation") AND

("Software Development" OR "Software engineering" OR "Web development") AND

("Code Analysis" OR "Code Review" OR "Code Inspection")

O recorte temporal de 2020 a 2025 fundamenta-se na rápida ascensão das arquiteturas baseadas em Transformers e *Large Language Models* (LLMs), que ampliaram a capacidade de geração de *feedback* em linguagem natural, substituindo mensagens estáticas por explicações contextualizadas.

Os volumes são detalhados na Tabela 1.

Tabela 1. Total de artigos retornados por base científica.

Base de Dados	Resultados
ACM Digital Library	3258
Scopus	3572
ScienceDirect	1765
Xplore	231
Total (Bases Primárias)	8826

Para mitigar eventuais lacunas e integrar a produção científica nacional de relevância, realizou-se uma busca adicional na SolLib (SBC OpenLib) e o processo de *snowballing*. A partir dessas fontes complementares, foram identificados 4 estudos adicionais (2 via SolLib e 2 via *snowballing*) que continham ferramentas alinhadas ao escopo da pesquisa.

Para a seleção dos estudos, foram aplicados os seguintes critérios:

Critérios de Inclusão (CI):

CI1: Artigos publicados em periódicos ou anais de conferências.

CI2: Publicações em língua inglesa.

CI3: Estudos que descrevem ferramentas funcionais e avaliadas empiricamente no contexto do ensino de programação.

Critérios de Exclusão (CE):

CE1: Literatura cinza, incluindo teses, dissertações, relatórios técnicos, capítulos de livros ou resumos expandidos.

CE2: Ferramentas de uso exclusivamente industrial/profissional. Embora identificadas na busca, estas foram excluídas por carecerem de suporte pedagógico e objetivos voltados ao aprendizado.

CE3: Artigos que apresentam apenas modelos teóricos ou arquiteturas sem implementação prática.

3.3 Processo de Seleção

A seleção dos artigos seguiu um fluxo estruturado de identificação, triagem e elegibilidade inspirado no modelo *Preferred Reporting Items for Systematic Reviews and Meta-Analyses* (PRISMA), conforme ilustrado na Figura 1.

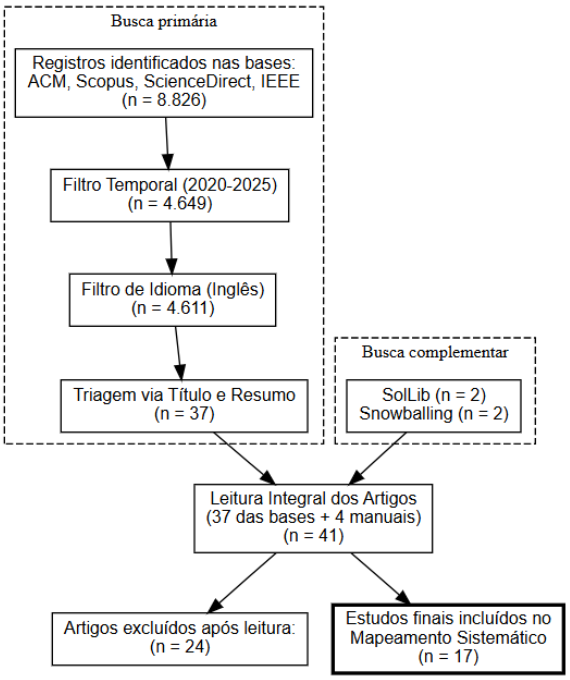

Figura 1. Fluxo de seleção dos artigos.

O processo foi conduzido em quatro fases:

Fase 1 - Identificação e filtragem automática: Inicialmente, foram recuperados 8.826 registros das bases de dados. Após a aplicação do filtro temporal (2020-2025) e a remoção de duplicatas, o volume foi reduzido para 4.611 artigos, priorizando tecnologias emergentes.

Fase 2 - Triagem por título e resumo: Realizou-se a análise manual da relevância técnica e pedagógica de cada registro. Em casos de ambiguidade nos resumos, a introdução dos trabalhos foi consultada. Esta etapa resultou na pré-seleção de 37 artigos para leitura integral.

Fase 3 - Integração de fontes: Ao conjunto inicial, foram integrados os 4 estudos identificados por meio das fontes complementares (2 via SolLib e 2 via *snowballing*), totalizando 41 candidatos para a análise de elegibilidade.

Fase 4 - Seleção final e auditoria: Os 41 artigos foram lidos na íntegra para verificar o cumprimento integral dos critérios de

inclusão (Seção 3.2). Ao final, consolidou-se o corpus de 17 ferramentas voltadas especificamente ao suporte educacional em programação. Para garantir a confiabilidade do processo e mitigar vieses de seleção, as etapas de inclusão e exclusão foram auditadas por um revisor sênior.

3.4 Extração e Categorização de Dados

Após a consolidação do corpus final de 17 ferramentas, procedeu-se à etapa de extração estruturada de dados. Essa fase consistiu no preenchimento de um formulário padronizado, estruturado com categorias previamente definidas com base nas Questões de Pesquisa (RQs) do estudo, com o objetivo de coletar evidências diretamente relacionadas aos objetivos investigados. A categorização foi realizada manualmente pelos autores a partir das descrições das ferramentas nos estudos primários. Em casos de ambiguidade, os artigos foram reavaliados conjuntamente até consenso. Foram encontradas três dimensões fundamentais para a categorização das ferramentas:

Tecnologia de Análise (Vinculada à RQ1): Identificação das abordagens utilizadas para o processamento do código e geração de *feedback*, classificadas em: análise estática tradicional, uso de *Large Language Models (LLMs)* ou abordagens híbridas.

Suporte à Qualidade (Vinculada à RQ2): Verificação da presença de orientações instrutivas baseadas em princípios de boas práticas de Engenharia de *Software* e legibilidade de código.

Qualidade Estrutural (Vinculada à RQ2): Avaliação da capacidade da ferramenta em oferecer suporte a aspectos relacionados à modularização, manutenibilidade e organização interna do código. Considerou-se suporte à qualidade estrutural a capacidade explícita da ferramenta em identificar problemas relacionados à modularização, duplicação, acoplamento excessivo ou *code smells* descritos pelos próprios autores.

Integração ao Fluxo do Estudante (Vinculada à RQ3): Foi analisado como a ferramenta se integra ao fluxo de desenvolvimento e interação do estudante, incluindo integração com IDEs, plataformas *Web*, repositórios Git e ambientes conversacionais, bem como o momento em que o *feedback* é disponibilizado.

4 Resultados e Discussão

Nesta seção apresentam-se os resultados obtidos a partir da análise das 17 ferramentas catalogadas. Os dados são discutidos baseando-se nas questões de pesquisa definidas no método, confrontando os trabalhos recuperados com as necessidades pedagógicas identificadas.

4.1 Panorama Geral das Ferramentas AFT

Após o processo de triagem e leitura completa, consolidou-se um conjunto de 17 ferramentas que oferecem suporte automatizado ao estudante. A Tabela 2 apresenta uma síntese das principais características identificadas, focando na tecnologia de geração de *feedback*, integração e suporte a boas práticas.

A coluna Ferramenta, apresenta a solução acompanhada de sua respectiva citação bibliográfica. Na sequência, a coluna Tecnologia especifica o motor de análise e geração de respostas, discriminando entre o uso de *Large Language Models* (LLMs), como GPT-3.5-turbo, GPT-4 e LLaMA 3, e métodos tradicionais de análise estática ou simbólica, a exemplo de AST, KLEE, Rascal e *linters* profissionais.

A terceira coluna, Fluxo, descreve a interface e o ambiente de integração para o usuário final, abrangendo plataformas *Web*, dispositivos móveis, integração nativa em IDEs e ecossistemas de desenvolvimento como GitHub e GitLab. Identifica-se também a existência de interfaces de chat, que permitem interação direta e dialógica entre o estudante e a ferramenta para o suporte em tempo real.

A coluna *Feedback* detalha a natureza qualitativa da saída entregue ao estudante, que pode variar entre revisões estruturais, análise de estilo, verificações de integridade, fornecimento de pseudocódigo ou explicações detalhadas em formato de prosa técnica, visando reduzir a carga cognitiva no processo de correção.

Complementarmente, a coluna Estrutural indica a capacidade da ferramenta em identificar problemas complexos relacionados à arquitetura, *code smells* e aos atributos de qualidade interna do código-fonte, elementos cruciais para a formação de desenvolvedores proficientes. O atributo Parcial nesta coluna refere-se a suporte indireto ou limitado a aspectos estruturais sem detecção explícita de *code smells*.

Por fim, a coluna Restrições mostra as estratégias pedagógicas implementadas para mediar o processo de ensino-aprendizagem, destacando mecanismos de controle como *guardrails*, voltados a evitar a entrega de soluções prontas, períodos de *cooldown*, que limitam a frequência de requisições para incentivar a reflexão, e a utilização de bases de conhecimento fixas para garantir que as respostas mantenham o foco pedagógico e o rigor acadêmico desejados.

Destaca-se que 41,17% (N=7) utilizam algum tipo de Large Language Model (LLM) ou Inteligência Artificial (IA) e quase todas, ou seja, 88,24% (N=15) proveem *feedback* imediato ao estudante. Esse suporte ocorre de formas distintas: em casos como o da ferramenta GitSeed [16], a devolutiva é apresentada logo após a submissão da atividade; já em soluções como o CodeAid [17], as orientações são fornecidas em tempo real, durante o desenvolvimento.

Adicionalmente, 35,29% (N=5) das ferramentas baseiam-se em interfaces de chat, permitindo interação direta via perguntas, ao passo que 23,53% (N=4) são integradas à IDE atuando diretamente no fluxo de trabalho do estudante

Tabela 2. Síntese das funcionalidades das AFTs analisadas.

Ferramenta	Tecnologia	Fluxo	*Feedback*	Estrutural	Restrição
APEJ [21]	AST	GitHub, Slack	Revisão estrutural, Revisão de estilo, Revisão de integridade, Pseudocódigo, geração de código.	Sim	*Code smells*, Estilo de código, integridade (*Security Risks*)
AI Knowledge Assistant [23]	GPT-4	*Web*/Kanban/ *Chat*	*Feedback* textual, Revisão estrutura, Geração de código.	Sim	*Guardrails*
AI-based OQAS [13]	OCLint, MLP	*Web*	*Feedback* textual, Revisão de estilo, Revisão de erros, Revisão estrutural	Sim	*Online Judge*
ART [19]	GPT 3.5-turbo	GitHub	*Feedback* textual, Revisão de estilo, Revisão de erros	Parcial	*Guardrails*
Auglets [4]	Algoritmo proprietário	*Web*	*Feedback* textual, Revisão de estilo, Revisão de erros, Revisão estrutural	Sim	Codificação guiada
C-BOT [22]	*Framework* Rasa	*Web/Chat*	*Quiz* sequencial e *Feedback* textual	Não identificado	Base de conhecimento fixa
CodeAID [17]	code-davinci-002, gpt-3.5-turbo	IDE/*Chat*	*Feedback* textual, Pseudocódigo	Parcial	*Guardrails*
CodeHelp [18]	gpt-3.5-turbo-0301, text-davinci-003	*Web/Chat*	*Feedback* textual	Parcial	*Guardrails*
CodeX [5]	LLaMA 3	Aplicação móvel/*Chat*	*Feedback* textual	Não identificado	Foco em TEA
CryptoTutor [25]	AST	Não especificado	Revisão de integridade, Geração de código, *Feedback* textual	Parcial	Segurança Criptográfica
GitSEED [16]	Bash e Python3	GitLab	*Feedback* textual	Parcial	*Cooldown*
Hyperstyle [14]	*Linters*	Stepik, JetBrains	*Feedback* textual, Revisão de estilo, Revisão de erros, Revisão estrutural	Limitado	*Guardrails*
KLC3 [11]	KLEE	VS Code, Git/GitHub, *Web*	*Feedback* textual, Revisão de estilo, Revisão de erros, Revisão estrutural	Sim	Solução referência
Personal Prof [6]	Rascal	Brightspace	*Feedback* textual, Revisão de estilo, Revisão de erros	Sim	*Cooldown*
Refactor Tutor [24]	*Framework* IDEAS	*Web*	*Feedback* textual, Revisão de estilo, Revisão de erros Árvore de Dicas	Sim	Base fixa
RTSF [12]	gpt-3.5-turbo	IDE	*Feedback* textual, Revisão de estilo, Revisão estrutural	Sim	*Cooldown*, *Guardrails*
SSDTutor [20]	AST	IDE	Revisão de estilo/integridade, *Feedback* textual, Revisão de erros	Sim	Criptografia

4.2 Análise Crítica e Discussão Qualitativa das Ferramentas

Esta seção constitui o núcleo analítico deste estudo, na qual as soluções selecionadas são confrontadas em termos de profundidade técnica e filosofia pedagógica.

4.2.1 Tensões Pedagógicas: *Scaffolding* x Automação

A análise detalhada das ferramentas revela uma divisão de visões sobre como a tecnologia deve auxiliar o estudante. Identificam-se dois paradigmas de interação: *design* de restrição (*guardrails*) e reparo automatizado.

***Design* de restrição (*Guardrails*):** Ferramentas como o CodeHelp [18] e o CodeAid [17] utilizam técnicas de *few-shot learning*, que consiste em fornecer ao modelo, dentro do próprio *prompt*, alguns exemplos (par de entrada e saída) de como ele deve se comportar. Ao contrário do *zero-shot* (onde se faz apenas uma pergunta direta), o *few-shot* permite que o *LLM* aprenda o padrão de resposta desejado por meio do contexto (*in-context learning*). O CodeHelp limita as respostas para que não incluam códigos, enquanto o CodeAid prioriza a geração de pseudocódigo e explicações linha por linha para incentivar o engajamento cognitivo. A ferramenta ART [19] reforça essa tendência ao fornecer revisões apenas em texto, não permitindo a geração de código nas respostas para evitar a programação por adivinhação, ou seja, por tentativa e erro e sem reflexão sobre o problema.

Reparo automatizado: No extremo oposto, soluções como o SSDTutor [20] e o Auglets [4] focam na correção direta do código. O SSDTutor gera o código necessário para corrigir vulnerabilidades em APIs criptográficas, enquanto o Auglets adota uma postura híbrida, removendo a liberdade de codificação para que o estudante pratique apenas operações de edição (excluir, duplicar e reordenar) voltadas a *anti-patterns* semânticos.

Implicações: Essa divisão evidencia uma tensão pedagógica central nas AFTs contemporâneas: enquanto algumas soluções priorizam o desenvolvimento da autonomia cognitiva do estudante, outras enfatizam a produtividade e a correção imediata, aproximando-se de assistentes automatizados de desenvolvimento. Esse tipo de abordagem pode reduzir oportunidades de reflexão ativa por parte do estudante, uma vez que entrega a solução.

4.2.2 Lacuna Entre Estilo e Qualidade Estrutural

Um dos principais achados deste estudo é a predominância de *feedback* voltado a aspectos superficiais do código na maioria das ferramentas educacionais.

Heurísticas de estilo e convenção: O *Personal Prof* [6] e o RTSF [12] priorizam *feedback* relacionado a convenções de nomenclatura e substituição de números mágicos por constantes nomeadas, com menor ênfase em aspectos estruturais do código. O Hyperstyle [14] segue uma abordagem semelhante, adaptando *linters* profissionais (como o da JetBrains) para fornecer mensagens pedagógicas sobre estilo e complexidade básica. A ferramenta auto descrita como *AI-based Online Code Quality Assessment System* (na Tabela 2 foi usado o nome do artigo, pois a ferramenta não foi nomeada pelos autores) [13] utiliza o OCLint para extrair métricas de redundância e tamanho de arquivos.

Baixo suporte à modularização: Como observado no estudo de base do *Personal Prof* [6], o *design* orientado a objetos e a arquitetura de *software* são os aspectos menos avaliados em sistemas automáticos. A ferramenta *Automating Programming Education in Java* (APEJ) [21] é a principal exceção técnica, utilizando Árvores de Sintaxe Abstrata (AST) para detectar indicadores complexos de *code smells,* como *God Class*, *Long Method* e *Data Clumps*. O *Refactor Tutor* [24] também busca preencher essa lacuna, oferecendo dicas para a melhoria de programas funcionalmente corretos.

Especialização e contextos diversos: O CodeX [5] foca em instruções objetivas e interações via *prompt* para auxiliar estudantes com TEA. O C-BOT [22] utiliza uma abordagem de *chatterbot* baseada em *quizzes* conceituais sobre a linguagem C. A ferramenta *AI Knowledge Assistant* (Zbot) [23] integra-se a ambientes *Kanban* para fornecer recomendações sensíveis ao contexto da tarefa. No escopo de cibersegurança e práticas seguras de programação, o CryptoTutor [25] atua na detecção automática de usos incorretos de *APIs* criptográficas na linguagem Java. A ferramenta auxilia na correção de programas vulneráveis gerando *patches* automatizados, fundamentados em modelos de segurança para seis tipos distintos de uso incorreto. Sob a ótica do suporte pedagógico, a solução destaca-se por suportar iterações do tipo perguntas e respostas (Q&A), permitindo que o estudante receba versões intermediárias e reenvie seu código continuamente até atingir uma convergência segura.

Implicações: Esse cenário sugere que atributos de qualidade estrutural continuam sendo mais difíceis de operacionalizar pedagogicamente em ambientes automatizados, especialmente quando comparados a métricas locais de estilo e legibilidade.

4.2.3 Arquiteturas de Disponibilização: IDE x Repositório

A forma como a ferramenta de suporte é instanciada determina a proximidade entre a autoria do código e a correção do erro.

Ecossistema centrado na IDE: Ferramentas como o CodeAid [17], o KLC3 [11], o SSDTutor [20] e o RTSF [12] operam como extensões diretas das IDEs (VS Code ou Eclipse). Essa arquitetura permite o fornecimento de *feedback* imediato por meio de sublinhados e pop-ups, o que é considerado útil por 80% dos usuários do KLC3 para a identificação de *bugs* sutis durante a edição [11]. A predominância de ferramentas especializadas em uma única linguagem sugere que parte das soluções educacionais ainda depende fortemente de ecossistemas específicos, dificultando a generalização do suporte pedagógico.

Dependência de repositórios (Git-based): Soluções como o GitSEED [16], a ART [19] e a APEJ [21] exigem que o estudante suba o seu código para o repositório (*push),* seja para o GitLab ou GitHub. No GitSEED, o retorno é enviado via arquivos *README* no próprio repositório, enquanto na APEJ, o sistema utiliza comentários automáticos nos arquivos online. Embora permitam integrações industriais como o Valgrind, estudantes apontam a falta de integração local como uma barreira de usabilidade.

Plataformas isoladas e *chatbots*: Ferramentas como o CodeHelp [18], o CodeX [5], o C-BOT [22] e o Refactor tutor [24] funcionam como ambientes *Web* ou agentes de conversação independentes. O Zbot [23], por exemplo, é incorporado a

interfaces de gerenciamento Kanban para prover assistência sensível ao contexto do projeto.

Adicionalmente, nota-se que a eficácia da entrega também esbarra na omissão de detalhes arquiteturais na literatura. Apesar de prover *feedback* em linguagem natural e em tempo real, trabalhos como o do CryptoTutor [25] apresentam lacunas quanto à sua implantação prática, não deixando claro o grau de integração com IDEs ou o evento exato que dispara o gatilho para a avaliação do código pelo sistema.

Implicações: Os resultados indicam um *trade-off* recorrente entre profundidade analítica e proximidade pedagógica: ferramentas mais integradas ao fluxo da IDE tendem a priorizar *feedback* leve e imediato, enquanto soluções com análises estruturais mais complexas frequentemente dependem de arquiteturas remotas ou baseadas em repositórios.

4.2.4 Controle de Frequência e o Fenômeno do *Cooldown*

Um achado singular deste estudo é o uso de mecanismos para mitigar a programação por adivinhação, ou seja, aquela na qual o estudante utiliza o *feedback* imediato como um substituto para o raciocínio crítico.

Períodos de resfriamento: Para forçar a reflexão, o GitSEED [16], o KLC3 [11], o Personal Prof [6] e o RTSF [12] implementam atrasos obrigatórios entre submissões, variando de 5 a 10 minutos.

Justificativa pedagógica: No Personal Prof [6], esse atraso de cinco minutos é deliberado para desencorajar que o estudante apenas reaja ao relatório de *feedback*, incentivando-o a desenvolver suas próprias habilidades de depuração antes de solicitar nova ajuda.

Implicações: Embora o *cooldown* estimule reflexão antes de novas submissões, mecanismos excessivamente restritivos podem introduzir fricção no processo de aprendizagem, especialmente em estudantes iniciantes que dependem de ciclos rápidos de experimentação.

4.2.5 Suporte a Múltiplas Linguagens de Programação

A capacidade de adaptação a diferentes tecnologias é um diferencial entre as ferramentas.

Agnosticismo de linguagem de programação: O GitSEED [16] destaca-se por ser independente da linguagem, operando com qualquer tecnologia suportada pelo pipeline do GitLab. O CodeHelp [18] também oferece suporte amplo, limitado apenas pela cobertura do LLM subjacente.

Especialização por linguagem: Diversas ferramentas, no entanto, focam em uma única linguagem de programação. As ferramentas Personal Prof [6], APEJ [21], Refactor tutor [24] e SSDTutor [20] operam no ecossistema Java. Já as ferramentas C-BOT [22], ART [19], OCLint System operam com o ambiente C/C++. O Hyperstyle [14] oferece uma cobertura intermediária em termos de linguagem de programação, suportando Python, Java, Kotlin e JavaScript.

Implicações: A predominância de ferramentas especializadas em linguagens específicas sugere que o suporte automatizado ainda depende fortemente de ecossistemas técnicos particulares, dificultando a criação de soluções pedagógicas amplamente reutilizáveis.

4.3 Discussão dos Resultados

A análise consolidada das 17 ferramentas permitiu: (i) gerar uma representação conceitual do cenário atual considerando a integração ao fluxo do estudante e a profundidade do *feedback*; (ii) sintetizar as respostas às questões de pesquisa deste estudo; e, (iii) identificar os desafios ao suporte automatizado do ensino de programação. Isso será detalhado nas próximas seções.

4.3.1 Representação Conceitual das AFTs

A Figura 2 apresenta o posicionamento conceitual das AFTs considerando no eixo X a profundidade do *feedback* oferecido ao estudante pela ferramenta e no eixo Y, o grau de integração ao fluxo do estudante. A representação favorece a compreensão do posicionamento das AFTs umas em relação às outras.

O quadrante mais maduro refere-se às ferramentas mais integradas ao fluxo do estudante e, ao mesmo tempo, que oferece o *feedback* mais detalhado em relação à qualidade estrutural. Neste caso, entre as ferramentas analisadas, a SSDTutor [20] apresentou o posicionamento mais próximo a este estado ideal.

De forma simular, o quadrante menos maduro, posicionado no canto inferior esquerdo da representação, refere-se às ferramentas com baixa integração ao fluxo do estudante e com *feedback* mais focado na parte de estilo e não na qualidade estrutural do código. Encontra-se neste quadrante a ferramenta CBOT [22].

4.3.2 Síntese das Questões de Pesquisa (RQs)

RQ1 - Quais tecnologias de análise e geração de *feedback* são utilizadas nas ferramentas de *feedback* automatizado atuais?

Observou-se uma transição progressiva para o uso de Large Language Models (LLMs) e Inteligência Artificial (IA), presente em 41,17% das ferramentas. Contudo, a tecnologia de análise estática via Árvore de Sintaxe Abstrata (AST), embora mais precisa para problemas estruturais, ainda é subutilizada no contexto educacional, aparecendo de forma robusta apenas em soluções específicas como a APEJ [21].

RQ2 - Como as ferramentas de *feedback* automatizado abordam atributos de qualidade e boas práticas de Engenharia de *Software*?

Há uma clara predominância de *feedback* voltado à legibilidade, estilo e convenções de padrões locais. O suporte a *design* e arquitetura (como *code smells*) representa uma lacuna recorrente nas ferramentas analisadas, sendo frequentemente negligenciado em favor da correção funcional.

RQ3: Como as ferramentas de *feedback* automatizado se integram ao fluxo de desenvolvimento e interação do estudante?

A integração com IDEs (VS Code, Eclipse) tende a favorecer maior proximidade entre o *feedback* e o processo de codificação. No entanto, ferramentas que realizam análises estruturais complexas tendem a ser *cloud-native* ou baseadas em Git, o que tende a introduzir barreiras de uso ao distanciar o *feedback* do fluxo de codificação. Foi constatado que apenas 4 ferramentas estão integradas à IDE.

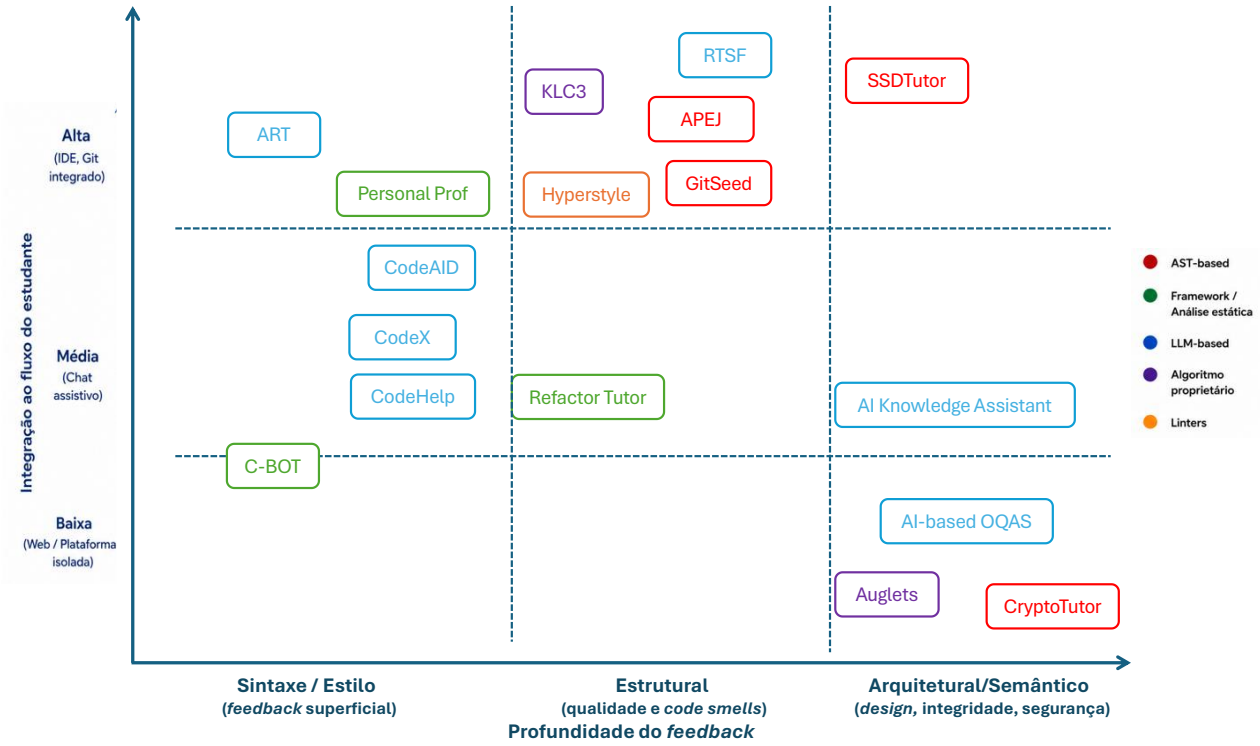

Figura 2. Mapeamento conceitual das AFTs em relação à integração ao fluxo do estudante e à profundidade do *feedback*.

4.3.3. Desafios Identificados

A partir do estudo realizado, emergem três desafios críticos para a área:

Desafio pedagógico da automação: Como equilibrar o uso de IAs que podem realizar o reparo automático do código com a necessidade de manter o engajamento cognitivo do estudante, evitando a dependência tecnológica?

Desafio da precisão vs. linguagem natural: Ferramentas baseadas apenas em LLM podem fornecer explicações fluidas, mas carecem da precisão de verificadores estáticos industriais para detectar problemas complexos de modularização.

Desafio da usabilidade oportuna: Implementar períodos de resfriamento (*cooldown*) é uma estratégia válida para evitar a tentativa e erro, mas ainda é necessário pesquisar como isso afeta a motivação do estudante no longo prazo. Paradas constantes podem impactar negativamente sua motivação para o uso da AFT.

5 Ameaças à Validade

Nesta seção, discutem-se as limitações do estudo realizado e as medidas de mitigação adotadas para garantir o rigor dos resultados.

Cobertura da busca e representatividade do corpus: Com o objetivo de ampliar a cobertura do cenário investigado, foram realizadas buscas em bases científicas consolidadas e complementadas por *snowballing*. Ainda assim, reconhece-se a possibilidade de omissão de ferramentas relevantes não publicadas em veículos científicos ou descritas apenas em documentação técnica e produtos comerciais.

Viés de seleção e escopo: Embora ferramentas industriais e comerciais emergentes tenham sido identificadas durante a busca, este estudo restringiu-se a soluções que apresentassem uma aplicação pedagógica explicitamente descrita. Ferramentas puramente industriais ou de análise estática genérica (como o OCLint [13] ou *linters* convencionais) foram consideradas apenas quando adaptadas para contextos educacionais. Assim, os resultados refletem prioritariamente o estado da pesquisa acadêmica sobre AFTs aplicadas ao ensino de programação.

Validade de construção: A categorização das ferramentas foi realizada com base em critérios técnicos explicitamente descritos nos estudos primários, buscando reduzir interpretações subjetivas sobre as funcionalidades oferecidas. A distinção entre *feedback* sintático e estrutural considerou evidências relacionadas à análise de modularização, detecção de *code smells*, uso de ASTs ou mecanismos equivalentes descritos pelos autores.

Validade externa: Os achados sobre a lacuna de suporte à qualidade estrutural são específicos ao contexto educacional investigado. É possível que, no contexto industrial, ferramentas como o CORE [26] ou *frameworks* como ISMELL [27] apresentem soluções mais avançadas para refatoração, porém, a transposição dessas tecnologias para o ambiente de ensino (onde objetivos pedagógicos diferem das prioridades normalmente presentes em ambientes industriais) ainda representa um desafio em aberto.

6 Conclusão

Este trabalho realizou um mapeamento da literatura estruturado sobre AFTs no ensino de programação, analisando 17 soluções sob as perspectivas de tecnologia, conteúdo pedagógico e infraestrutura de acesso. Os resultados revelam um cenário de tendência ao uso de LLMs e IA (41,17%), que embora tenham trazido avanços na naturalidade das explicações, ainda apresentam limitações no suporte ao ensino do desenvolvimento de *software*.

Uma das principais contribuições deste estudo é a identificação de lacunas relacionadas ao suporte à qualidade estrutural no ensino de programação, especificamente no ensino de *design* e arquitetura (identificação de *code smells*).

A análise indica que a maioria das ferramentas foca em conformidade sintática ou de estilo, deixando o estudante sem suporte para lidar com a dívida técnica e a modularização, aspectos que são fundamentais para a transição de um programador iniciante para um desenvolvedor profissional.

Os resultados deste estudo sugerem que a evolução das AFTs não depende apenas de avanços em modelos de linguagem ou infraestrutura de análise, mas principalmente da capacidade de transformar mecanismos automatizados de revisão em instrumentos pedagogicamente orientados ao desenvolvimento de competências estruturais em Engenharia de *Software*.

As evidências coletadas neste mapeamento sugerem caminhos para o avanço da área, os quais já fundamentam o desenvolvimento de uma nova abordagem de suporte ao ensino de desenvolvimento de *software* por parte dos autores deste trabalho. Entre as frentes prioritárias, destacam-se as listadas a seguir.

Apoio à qualidade estrutural na IDE: Investigar a viabilidade de integrar análises de código diretamente no ambiente de desenvolvimento (IDE) do estudante, buscando apoiá-lo no momento da autoria.

Agentes de tutoria pedagógica: Explorar o uso de arquiteturas baseadas em agentes inteligentes para fornecer *feedbacks* que estimulem a reflexão crítica, em vez de focar apenas na correção automatizada do código.

Avaliação sem soluções de referência: Investigar métodos de análise capazes de operar sem depender de soluções previamente definidas, viabilizando suporte automatizado em atividades de programação com múltiplas estratégias válidas de resolução.

Impacto de Mecanismos de Controle: Analisar como estratégias de fluxo (como o *cooldown*) influenciam a autonomia do estudante e a retenção de longo prazo de conceitos de qualidade.

Disponibilidade de Artefatos

Para apoiar a transparência e facilitar a replicação, o protocolo do estudo, os instrumentos de coleta de dados e os materiais suplementares estão disponíveis em: https://doi.org/10.5281/zenodo.21421092

Declaração de Uso de Inteligência Artificial Generativa

Inteligência artificial generativa foi utilizada para auxiliar na edição do idioma, melhorar a clareza e a legibilidade, refinar a organização do manuscrito e fornecer *feedback* editorial durante a preparação do trabalho. Todas as decisões de desenho da pesquisa, análise de dados, interpretação dos resultados e conclusões científicas foram conduzidas, verificadas e revisadas criticamente pelos autores, que assumem total responsabilidade pelo conteúdo deste artigo.

Referências

[1] Hironori Washizaki, ed. 2024. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0. IEEE Computer Society, Los Alamitos, CA, USA.

[2] Robert C. Martin. 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

[3] Ian Sommerville. 2011. Engenharia de Software (9ª ed.). Pearson Prentice Hall.

[4] Amruth N. Kumar. 2024. Auglets: Intelligent Tutors for Learning Good Coding Practices by Solving Refactoring Problems. In Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual 2024). Association for Computing Machinery, New York, NY, USA, 95-101. https://doi.org/10.1145/3649165.3690119.

[5] Vitor Norton, Fabrizio Honda, Marcela Pessoa, and Fernanda Pires. 2024. CodeX: auxiliando estudantes com TEA em programação por meio de um Ambiente Virtual de Aprendizagem (AVA) integrado à um LLM. In Anais Estendidos do XIII Congresso Brasileiro de Informática na Educação, novembro 04, 2024, Rio de Janeiro/RJ, Brasil. SBC, Porto Alegre, Brasil, 155-158. DOI: https://doi.org/10.5753/cbie_estendido.2024.244218.

[6] Markus Klinik, Pieter Koopman, and Rick van der Wal. 2022. Personal Prof: Automatic Code Review for Java Assignments. In Proceedings of the 10th Computer Science Education Research Conference (CSERC '21). Association for Computing Machinery, New York, NY, USA, 31–38. https://doi.org/10.1145/3507923.3507930.

[7] Martin Fowler. 2018. Refactoring: Improving the Design of Existing Code (2nd ed.). Addison-Wesley Professional.

[8] John Hattie and Helen Timperley. 2007. The Power of Feedback. Review of Educational Research 77, 1 (March 2007), 81–112. https://doi.org/10.3102/003465430298487.

[9] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. ACM Trans. Comput. Educ. 19, 1, Article 3 (March 2019), 43 pages. https://doi.org/10.1145/3231711.

[10] Susanne Narciss. 2008. Feedback Strategies for Interactive Learning Tasks. In Handbook of Research on Educational Communications and Technology, J. Michael Spector, M. David Merrill, Jeroen van Merriënboer, and Marcy P. Driscoll (Eds.). Lawrence Erlbaum Associates, 125–144.

[11] Z. Liu, T. Liu, Q. Li, W. Luo and S. S. Lumetta. 2021. End-to-End Automation of Feedback on Student Assembly Programs. 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 18-29, doi: 10.1109/ASE51524.2021.9678837.

[12] Juliette Woodrow, Ali Malik, and Chris Piech. 2024. AI Teaches the Art of Elegant Coding: Timely, Fair, and Helpful Style Feedback in a Global Course. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024). Association for Computing Machinery, New York, NY, USA, 1442–1448. https://doi.org/10.1145/3626252.3630773.

[13] S. Yi, Y. Yu and J. Wu. AI-based Online Code Quality Assessment System. 2024. 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE), Hangzhou, China, 2024, pp. 659-663, doi: 10.1109/CBASE64041.2024.10824380.

[14] Anastasiia Birillo, Ilya Vlasov, Artyom Burylov, Vitalii Selishchev, Artyom Goncharov, Elena Tikhomirova, Nikolay Vyahhi, and Timofey Bryksin. 2022. Hyperstyle: A Tool for Assessing the Code Quality of Solutions to Programming Assignments. In Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1 (SIGCSE 2022), Vol. 1. Association for Computing Machinery, New York, NY, USA, 307–313. https://doi.org/10.1145/3478431.3499294.

[15] Barbara Kitchenham and Stuart Charters, 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-01. School of Computer Science and Mathematics, Keele University. URL: https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf.

[16] Pedro Orvalho, Mikolás Janota, and Vasco Manquinho. 2024. GitSEED: A Git-backed Automated Assessment Tool for Software Engineering and Programming Education. In Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual 2024). Association for

Computing Machinery, New York, NY, USA, 165–171. https://doi.org/10.1145/3649165.3690106.

[17] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI '24). Association for Computing Machinery, New York, NY, USA, Article 650, 1–20. https://doi.org/10.1145/3613904.3642773.

[18] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2024. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. In Proceedings of the 23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23). Association for Computing Machinery, New York, NY, USA, Article 8, 1–11. https://doi.org/10.1145/3631802.3631830.

[19] Aaron S. Crandall, Gina Sprint, and Bryan Fischer. 2023. Generative Pre-Trained Transformer (GPT) Models as a Code Review Feedback Tool in Computer Science Programs. J. Comput. Sci. Coll. 39, 1 (October 2023), 38–47. DOI: https://dl.acm.org/doi/10.5555/3636517.3636522.

[20] Dip Kiran Pradhan Newar, Rui Zhao, Harvey Siy, Leen-Kiat Soh and Myoungkyu Song. 2023. SSDTutor: A feedback-driven intelligent tutoring system for secure software development. Science of Computer Programming 227 (2023), 102933. DOI: https://doi.org/10.1016/j.scico.2023.102933.

[21] Matthew Beattie, Moira Watson, Desmond Greer, Bee-Yen Toh and Zheng Li. 2025. Code-Review-as-an-Educational-Service: A tool for Java code review in programming education. SoftwareX 29 (2025), 102048. DOI: https://doi.org/10.1016/j.softx.2025.102048.

[22] João Eduardo Soares and Larissa de Freitas. 2022. C-BOT: Um protótipo de chatterbot para o ensino de programação. In Anais do XXXIII Simpósio Brasileiro de Informática na Educação, novembro 16, 2022, Manaus, Brasil. SBC, Porto Alegre, Brasil, 1151-1162. DOI: https://doi.org/10.5753/sbie.2022.225711.

[23] A. Neyem, L. A. González, M. Mendoza, J. P. S. Alcocer, L. Centellas and C. Paredes, "Toward an AI Knowledge Assistant for Context-Aware Learning Experiences in Software Capstone Project Development," in IEEE Transactions on Learning Technologies, vol. 17, pp. 1599-1614, 2024, doi: 10.1109/TLT.2024.3396735

[24] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2021. A Tutoring System to Learn Code Refactoring. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21). Association for Computing Machinery, New York, NY, USA, 562–568. https://doi.org/10.1145/3408877.3432526.

[25] Larry Singleton, Rui Zhao, Myoungkyu Song, and Harvey Siy. 2020. CryptoTutor: Teaching Secure Coding Practices through Misuse Pattern Detection. In Proceedings of the 21st Annual Conference on Information Technology Education (SIGITE '20). Association for Computing Machinery, New York, NY, USA, 403–408. https://doi.org/10.1145/3368308.3415419.

[26] Nalin Wadhwa, Jui Pradhan, Atharv Sonwane, Surya Prakash Sahu, Nagarajan Natarajan, Aditya Kanade, Suresh Parthasarathy, and Sriram Rajamani. 2024. CORE: Resolving Code Quality Issues using LLMs. Proc. ACM Softw. Eng. 1, FSE, Article 36 (July 2024), 23 pages. https://doi.org/10.1145/3643762

[27] Di Wu, Fangwen Mu, Lin Shi, Zhaoqiang Guo, Kui Liu, Weiguang Zhuang, Yuqi Zhong, and Li Zhang. 2024. ISMELL: Assembling LLMs with Expert Toolsets for Code Smell Detection and Refactoring. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24). Association for Computing Machinery, New York, NY, USA, 1345–1357. https://doi.org/10.1145/3691620.3695508.