UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO BACHARELADO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

JEFERSSON FRANCISCO GAMELEIRA PEREIRA

UM ESTUDO SOBRE O USO DE TESTES PARA ENSINO DE PROGRAMAÇÃO INTRODUTÓRIA

PAU DOS FERROS
2026

JEFERSSON FRANCISCO GAMELEIRA PEREIRA

UM ESTUDO SOBRE O USO DE TESTES PARA ENSINO DE PROGRAMAÇÃO INTRODUTÓRIA

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Orientador: José Ferdiandy Silva Chagas, Prof. Me

PAU DOS FERROS

2026

© Todos os direitos estão reservados a Universidade Federal Rural do Semi-Árido. O conteúdo desta obra é de inteira responsabilidade do (a) autor (a), sendo o mesmo, passível de sanções administrativas ou penais, caso sejam infringidas as leis que regulamentam a Propriedade Intelectual, respectivamente, Patentes: Lei n° 9.279/1996 e Direitos Autorais: Lei n° 9.610/1998. O conteúdo desta obra tomar-se-á de domínio público após a data de defesa e homologação da sua respectiva ata. A mesma poderá servir de base literária para novas pesquisas, desde que a obra e seu (a) respectivo (a) autor (a) sejam devidamente citados e mencionados os seus créditos bibliográficos.

P436e Pereira, Jefersson Francisco Gameleira.
Um estudo sobre o uso de testes para ensino de programação introdutória / Jefersson Francisco Gameleira Pereira. - 2026.
69 f. : il.

Orientador: José Ferdiandy Silva Chagas.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2026.

1. Desenvolvimento orientado a testes. 2. Educação em computação. 3. Programação introdutória. 4. Pensamento computacional. 5. Testes de software. I. Chagas, José Ferdiandy Silva, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

JEFERSSON FRANCISCO GAMELEIRA PEREIRA

UM ESTUDO SOBRE O USO DE TESTES PARA ENSINO DE PROGRAMAÇÃO INTRODUTÓRIA

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Defendida em: 26/06/2026

BANCA EXAMINADORA

José Ferdiandy Silva Chagas, Prof. Me (UFERSA)
Presidente

Reudismam Rolim de Sousa, Prof. Dr. (UFERSA)
Membro Examinador

Huliane Medeiros da Silva, Prof. Dra. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço, de forma especial, à minha família, que sempre foi minha base de apoio durante toda esta jornada. Em especial, à minha mãe, Maria Laura Pereira da Silva, pelo amor, dedicação, incentivo e por sempre acreditar em meu potencial.

Também presto minha homenagem e eterna gratidão ao meu falecido pai, José Peixoto da Silva Neto, cuja memória, ensinamentos e valores permanecem presentes em minha vida, servindo de inspiração para minhas conquistas e para a pessoa que me tornei.

EPÍGRAFE

"A educação não transforma o mundo. A educação muda as pessoas. Pessoas transformam o mundo."

(Paulo Freire)

RESUMO

O ensino introdutório de programação constitui um dos principais pilares da formação em Computação, embora ainda enfrente desafios relacionados ao desenvolvimento do raciocínio lógico, à compreensão de problemas e à construção de soluções computacionais por estudantes iniciantes. Paralelamente, práticas amplamente utilizadas na indústria de software, como o Desenvolvimento Orientado por Testes (TDD), têm demonstrado potencial para melhorar a qualidade do código e promover uma abordagem mais estruturada no desenvolvimento de sistemas. Nesse contexto, este trabalho teve como objetivo analisar a aplicabilidade da metodologia TDD no ensino introdutório de programação, bem como investigar as percepções sobre seus potenciais benefícios, considerando evidências identificadas na literatura e a percepção de estudantes, docentes e profissionais da área de tecnologia. Para tanto, realizou-se uma revisão da literatura sobre o uso de testes de software e TDD em contextos educacionais, complementada por uma pesquisa do tipo *survey*, aplicada a estudantes, docentes e profissionais da área de tecnologia, totalizando 32 participantes. A pesquisa configurou-se como uma abordagem quantitativa e descritiva. Os resultados indicaram que a maior parte dos participantes enfrentou dificuldades durante o aprendizado inicial de programação, e que o contato com testes de software nas disciplinas introdutórias ainda é limitado. Por outro lado, observou-se uma percepção favorável dos participantes quanto aos potenciais benefícios do TDD, especialmente no que diz respeito à compreensão de requisitos, à organização das soluções, à qualidade do código e ao desenvolvimento do raciocínio lógico. Com base nos resultados obtidos, propôs-se uma abordagem gradual para a implementação do TDD no ensino introdutório de programação. Conclui-se que, à luz das evidências identificadas na literatura e das percepções obtidas por meio do survey, a metodologia apresenta potencial para contribuir com o processo de aprendizagem e aproximar a formação acadêmica das práticas contemporâneas da indústria de software.

Palavras-chave: desenvolvimento orientado a testes; educação em computação; programação introdutória; pensamento computacional; testes de software.

ABSTRACT

Introductory programming instruction is one of the main pillars of computer science education, although it still faces challenges related to the development of logical reasoning, problem understanding, and the construction of computational solutions by beginner students. At the same time, practices widely used in the software industry, such as Test-Driven Development (TDD), have shown potential for improving code quality and promoting a more structured approach to systems development. In this context, the objective of this study was to analyze the applicability of the TDD methodology in introductory programming education, as well as to investigate perceptions regarding its potential benefits, considering evidence identified in the literature and the perspectives of students, faculty, and professionals in the technology field. To this end, a literature review was conducted on the use of software testing and TDD in educational contexts, supplemented by a survey administered to students, faculty, and technology professionals, totaling 32 participants. The study employed a quantitative and descriptive approach. The results indicated that most participants faced difficulties during their initial programming learning experience and that exposure to software testing in introductory courses remains limited. On the other hand, participants expressed a favorable perception of the potential benefits of TDD, especially regarding understanding requirements, organizing solutions, code quality, and the development of logical reasoning. Based on the results obtained, a gradual approach to implementing TDD in introductory programming education was proposed. It is concluded that, in light of the evidence identified in the literature and the perceptions obtained through the survey, the methodology has the potential to contribute to the learning process and bring academic training closer to contemporary practices in the software industry.

Keywords: test-driven development; computer science education, introductory programming. computational thinking; software testing.

LISTA DE TABELAS

Tabela 1 – Comparação entre os trabalhos relacionados 29
Tabela 2 – Exemplo de aplicação da abordagem proposta 49
Tabela 3 – Relação entre as questões do questionário e os objetivos específicos da pesquisa 67

LISTA DE FIGURAS

Figura 1 – Tempo de experiência com programação. 38
Figura 2 – Linguagens de programação mais utilizadas. 39
Figura 3 – Dificuldade inicial no contato com programação. 40
Figura 4 – Fluxo de Ensino de programação com TDD 47

SUMÁRIO

1 INTRODUÇÃO 12

1.1 Objetivos 14

1.1.1 Objetivos específicos 14

1.2 Justificativa 15

1.3 Organização do trabalho 17

2 REFERENCIAL TEÓRICO 18

2.1 Informática na Educação 18

2.2 Pensamento Computacional 20

2.3 Metodologia Ágil 21

2.4 Qualidade de Software 22

3 TRABALHOS RELACIONADOS 26

3.1 Test-Driven Development: Challenges and Recommendations for Trainers and Educators 26

3.2 Qual tipo de feedback ajuda os estudantes novatos em programação? Um estudo exploratório com múltiplos feedbacks em um curso introdutório de Python 27

3.3 Desenvolvimento baseado em testes com aplicação de critérios de teste no ensino introdutório de computação 27

3.3.1 Síntese Comparativa dos Trabalhos Relacionados 28

3.3.2 Síntese Comparativa dos Trabalhos Relacionados 28

3.3.3 Síntese Comparativa dos Trabalhos Relacionados 28

4 METODOLOGIA 30

4.1 Levantamento Bibliográfico 30

4.2 Survey 31

4.3 Coleta de dados 32

4.4 Análise de dados 33

5 RESULTADOS 34

5.1 Levantamento Bibliográfico 34

5.2 Aplicação de Survey 36

5.2.1 Coleta de dados 37

5.2.2 Análise de Dados 38

5.2.2.1 Perfil dos participantes 38

5.2.2.2 Dificuldades iniciais com programação 39

5.2.2.3 Contato inicial com testes de software 39

5.2.2.4 Uso de testes automatizados 40

5.2.2.5 Familiaridade com o TDD 41

5.2.2.6 Aplicação prática do TDD 42

5.2.2.7 Percepção sobre o impacto do TDD 42

5.2.2.8 Contribuição do TDD para compreensão e raciocínio lógico 43

5.2.3 Discussão da Pesquisa 43

5.3 Proposta da abordagem baseada em TDD 46

5.3.1 Exemplo de aplicação da proposta em uma disciplina 49

5.3.1.1 Exemplo prático de aplicação do ciclo TDD 49

6 CONCLUSÕES E TRABALHOS FUTUROS 53

6.1 Limitações da Pesquisa 54

6.2 Trabalhos Futuros 54

APÊNDICE A QUESTIONÁRIO 58

1 INTRODUÇÃO

O ensino de programação introdutória desempenha um papel central na formação de profissionais da área de tecnologia, sendo responsável pelo desenvolvimento de competências relacionadas ao raciocínio lógico, resolução de problemas e pensamento computacional. Segundo Brackmann (2023), o pensamento computacional envolve a capacidade de utilizar fundamentos da ciência da computação para formular problemas e construir soluções de maneira estruturada e eficiente. No entanto, diversos estudos apontam que estudantes iniciantes frequentemente enfrentam dificuldades nesse processo de aprendizagem, especialmente em atividades que exigem abstração, interpretação de requisitos e construção incremental de soluções computacionais, fatores que contribuem para elevados índices de dificuldade e retenção em disciplinas introdutórias de programação (Araujo; Flach; Bittencourt, 2024; Feitosa; Silva; Fabri, 2021).

Paralelamente, a evolução da indústria de software impulsionou a adoção de práticas voltadas à melhoria da qualidade do código e à redução de defeitos ao longo do ciclo de desenvolvimento. Entre essas práticas, destaca-se o *Test-Driven Development* (TDD), metodologia difundida por Kent Beck no contexto do desenvolvimento ágil e baseada na criação de testes automatizados antes da implementação do código. O TDD propõe um processo iterativo estruturado em ciclos curtos de especificação, desenvolvimento e refatoração, com o objetivo de aumentar a confiabilidade do software, facilitar sua manutenção e promover uma construção incremental das soluções desenvolvidas (Beck, 2003; Martin, 2019).

Desta forma, a programação é uma competência indispensável para profissionais da área de tecnologia, sendo considerada uma das bases para a atuação em diferentes setores (Dargains; Sampaio, 2020). No entanto, durante o ensino introdutório da programação em contexto acadêmico, nota-se a existência de lacunas significativas que podem prejudicar o aprendizado de primeiro contato com a programação.

Em geral, as ementas das disciplinas iniciais enfatizam a construção de programas e o desenvolvimento do raciocínio lógico, porém deixam de lado práticas fundamentais e essenciais de qualidade de software, como a criação e execução de testes (Lopes; Fürkotter, 2016).

Além da carência de práticas de testes, observa-se que, mesmo quando presentes, os mecanismos de *feedback* oferecidos aos discentes costumam ser limitados. Estudos indicam que mensagens de erro convencionais frequentemente apresentam baixa eficácia para programadores iniciantes, e que a integração de testes unitários com sistemas de *feedback* imediato pode melhorar significativamente tanto a compreensão conceitual quanto a capacidade de correção de

erros (Araujo; Flach; Bittencourt, 2024).

No entanto, a efetividade desses recursos depende criticamente da forma como são implementados pedagogicamente e adotados pelos alunos. A simples disponibilização de ferramentas não garante a assimilação adequada de conceitos ou o desenvolvimento de habilidades de depuração. Portanto, é essencial que a incorporação de tais mecanismos seja acompanhada de orientação didática e integrada a um contexto de aprendizagem que incentive a reflexão e a melhoria contínua (Martins, 2019).

A defasagem pedagógica observada tende a resultar na formação de programadores que, mesmo dominando a sintaxe e compreendendo conceitos básicos de qualidade, não cultivam o hábito de empregar mecanismos sistemáticos de validação, comprometendo a garantia de qualidade e robustez do código (Martins, 2019).

A ausência de práticas adequadas de testes de software tem se mostrado um fator determinante em diversas falhas de grande impacto técnico, financeiro e social. Um exemplo notável ocorreu em outubro de 2021, quando uma atualização incorretamente testada na infraestrutura da empresa Meta resultou em uma interrupção global dos seus serviços por aproximadamente seis horas. O incidente ocasionou prejuízos bilionários em receitas publicitárias e afetou um amplo ecossistema de empresas e usuários que dependem das plataformas do conglomerado (Kobak, 2024).

De forma semelhante, o caso do sistema MCAS (*Maneuvering Characteristics Augmentation System*) da Boeing evidencia as consequências críticas da falta de testes abrangentes. O software apresentou falhas devido a leituras incorretas do sensor de ângulo de ataque, resultantes da ausência de testes em cenários extremos de falhas. Essa negligência contribuiu para dois acidentes fatais, totalizando 346 vítimas, além de prejuízos estimados em mais de 20 bilhões de dólares em indenizações, multas e paralisações operacionais (Administration, 2020).

Conforme descrito em estudo publicado pela *Testing Company*, a falta de testes estruturados e sistemáticos durante o ciclo de desenvolvimento leva à não detecção de erros críticos, ao aumento dos custos de manutenção e à instabilidade nos produtos entregues. Organizações que negligenciam testes automatizados e regressivos enfrentam riscos significativos de introduzir falhas em módulos essenciais dos sistemas. Ademais, a ausência de métricas de qualidade bem definidas dificulta a avaliação da saúde do projeto e compromete a confiabilidade das entregas (Baumgartner, 2025).

Desde a publicação do Manifesto Ágil em 2001, observa-se uma diversidade de per-

cepções sobre sua adoção nas organizações: enquanto algumas o compreendem como uma abordagem voltada ao aprimoramento contínuo dos processos de desenvolvimento e à adaptação às mudanças, outras tendem a interpretá-lo como uma mudança de caráter predominantemente burocrático, focada apenas na substituição de práticas e estruturas formais, sem incorporar seus princípios culturais e colaborativos (Conboy, 2009). Nesse contexto, surge a questão sobre até que ponto essa transformação pode ser conduzida de forma eficaz e quais suas implicações para a aprendizagem e o desempenho de novos profissionais na área de desenvolvimento de software. Diante desse panorama, pode-se afirmar que as instituições de ensino enfrentam desafios na adaptação de seus métodos pedagógicos às práticas atualmente adotadas no mercado de trabalho (Rocha *et al.*, 2021).

Mediante o exposto, surge a questão central deste estudo: Quais são as oportunidades e os desafios associados à aplicação do TDD no ensino introdutório de programação?

Nesse contexto, este trabalho tem como objetivo analisar a aplicabilidade e os potenciais benefícios da integração do TDD no ensino introdutório de programação, considerando aspectos relacionados à qualidade do código, ao desenvolvimento do raciocínio lógico e à aproximação entre a formação acadêmica e as práticas do mercado de tecnologia. Para isso, foi realizada uma revisão da literatura e uma pesquisa do tipo *survey* com estudantes, docentes e profissionais da área de tecnologia.

1.1 Objetivos

O objetivo principal desse trabalho é investigar a aplicabilidade e os benefícios potenciais da integração da metodologia TDD no contexto do ensino introdutório de programação. O estudo visa analisar a percepção de profissionais e estudantes da área de tecnologia sobre essa integração, com ênfase em como o TDD pode influenciar a qualidade do código, o desenvolvimento do raciocínio lógico e a preparação dos alunos para as práticas do mercado de trabalho.

1.1.1 Objetivos específicos

Para atingir o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- Realizar uma revisão da literatura sobre a aplicação da metodologia TDD, com foco no ensino de programação introdutória e os potenciais benefícios e desafios descritos na

literatura.

- Conduzir uma pesquisa do tipo *survey* com estudantes e profissionais da área de tecnologia, a fim de investigar percepções, níveis de conhecimento e experiências relacionadas ao uso do TDD.
- Analisar os dados coletados na *survey*, buscando identificar padrões, percepções sobre potenciais benefícios, desafios e tendências quanto à aplicação do TDD no contexto educacional e profissional.
- Propor uma abordagem para a utilização do TDD no ensino introdutório de programação, fundamentada nos resultados obtidos e nas evidências identificadas na literatura.

1.2 Justificativa

O ensino de programação introdutória é um pilar crítico para a formação de profissionais de tecnologia, área em constante expansão na sociedade digital. No entanto, a persistência de metodologias tradicionais, focadas predominantemente na sintaxe, gera uma lacuna formativa preocupante: a desconexão entre o conhecimento acadêmico e as competências práticas exigidas pela indústria de software, tais como a produção de código confiável e a aplicação sistemática de testes (Lopes; Fürkotter, 2016; Martins, 2019).

Diante dessa situação, essa pesquisa é importante não só para identificar o problema, mas também pelo potencial de apresentar uma solução concreta para a modernização do ensino. O estudo sobre a inclusão do TDD na fase inicial vai além de apenas apontar uma dificuldade; ele busca analisar alternativas práticas, apoiadas em estudos e experiências pedagógicas já registradas.

As principais contribuições e a relevância deste estudo são:

1. **Potencial de Modernização Pedagógica:** Esta pesquisa busca maneiras de aprimorar o ensino de computação, investigando a integração de uma prática industrial consagrada, o TDD, no contexto educacional fundamental. Isso representa um avanço em relação ao modelo tradicional, propondo uma metodologia ativa que coloca o aluno como protagonista do processo de aprendizagem, incentivando a validação contínua e o pensamento crítico desde o primeiro contato com a programação.
2. **Síntese de Evidências para Educadores**: O estudo realizado por Feitosa, Silva e Fabri em 2021, indica os benefícios associados ao uso de testes no ensino introdutório de programação. Esta pesquisa busca organizar e analisar criticamente essas evidências,

oferecendo aos professores e coordenadores uma base sólida e confiável para pensar em mudanças nas disciplinas e nas metodologias de ensino.

3. **Abordagem Prática e Contextualizada**: A utilização da linguagem Python e de ferramentas como o Pytest conecta a proposta a um ecossistema acessível e amplamente utilizado, tanto nas instituições de ensino quanto na indústria. Isso aumenta a viabilidade e o impacto prático da proposta, fornecendo um exemplo tangível de como a teoria pode ser aplicada.
4. **Aproximação entre Instituições de Ensino e Mercado de Trabalho**: Este estudo busca contribuir para a aproximação entre as instituições de ensino e as práticas adotadas pela indústria de software. Ao investigar a aplicabilidade do TDD no ensino introdutório de programação, a pesquisa reúne evidências da literatura e percepções de estudantes, docentes e profissionais da área de tecnologia, oferecendo subsídios para que educadores possam refletir sobre a incorporação gradual de práticas de qualidade de software no processo formativo. Espera-se, assim, contribuir para uma formação mais alinhada às competências valorizadas pelo mercado de trabalho.

Estudos, como o de Zangara et al., apontam que a implementação do TDD em disciplinas de programação está associada a resultados favoráveis, tais como o aumento do engajamento dos estudantes, a melhoria na compreensão dos conceitos de testes e desenvolvimento incremental, além de fomentar mudanças positivas na cultura de programação dos alunos. Além do contexto acadêmico, profissionais da área de tecnologia que adotam o TDD em seus processos de desenvolvimento relatam resultados positivos como a elevação da qualidade do código, maior segurança na realização de refatorações, detecção precoce de falhas e maior organização na implementação dos requisitos. Esses resultados evidenciam a importância de estratégias voltadas à formação e ao suporte ao corpo docente para diminuir resistências e ampliar a adoção de práticas inovadoras nos cursos introdutórios de computação, aproximando o ambiente acadêmico das práticas consolidadas na indústria de software. Dessa forma, este estudo busca oferecer subsídios teóricos e práticos para professores, gestores e instituições, fortalecendo o papel do docente como agente central na transformação do ensino de programação.

Ressalta-se que a utilização de testes no contexto educacional não é um conceito inteiramente novo. Atualmente, plataformas amplamente empregadas no ensino de programação, como juízes online e ambientes de correção automática, já utilizam mecanismos baseados em casos de teste para validar as soluções desenvolvidas pelos estudantes. Nesses sistemas, normalmente, são disponibilizados problemas com entradas e saídas esperadas, enquanto conjuntos adicionais de

testes permanecem ocultos a fim de verificar a robustez da solução. O estudante desenvolve seu programa de forma livre e, ao final, submete o código para avaliação, recebendo como retorno a aprovação ou reprovação com base na execução dos testes. Entretanto, essa abordagem difere da metodologia do TDD. Enquanto os juízes online utilizam os testes como instrumento de validação posterior à construção da solução, o TDD propõe que os testes participem ativamente do processo de desenvolvimento desde o início, orientando a implementação, aprimorando a compreensão dos requisitos e fomentando ciclos contínuos de validação e melhoria. Nesse sentido, a proposta deste trabalho busca aproveitar a familiaridade dos estudantes com mecanismos de avaliação automatizada para introduzir gradualmente os princípios do desenvolvimento orientado por testes, convertendo os testes de uma ferramenta exclusivamente avaliativa em um recurso de suporte ao aprendizado e à construção do raciocínio computacional.

1.3 Organização do trabalho

O trabalho está organizado da seguinte forma: o Capítulo 2 apresenta o referencial teórico; o Capítulo 3 aborda os trabalhos relacionados; o Capítulo 4 descreve a metodologia utilizada; o Capítulo 5 apresenta e discute os resultados obtidos; e, por fim, o Capítulo 6 apresenta as conclusões e sugestões para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão apresentados os principais conceitos teóricos que subsidiaram a revisão da literatura e serviram de base para o desenvolvimento deste trabalho. O objetivo é oferecer um embasamento consistente que permita compreender de forma ampla os temas relacionados à aplicação de metodologias de desenvolvimento e práticas de testes no contexto educacional, especialmente no ensino de programação introdutória.

Na seção 2.1, será abordado o conceito de Informática na Educação, discutindo sua importância como ferramenta pedagógica, sua evolução histórica e as formas como tem contribuído para o aprimoramento dos processos de ensino e aprendizagem.

Na seção 2.2, será explorado o Pensamento Computacional, destacando suas principais habilidades cognitivas — como decomposição, abstração e reconhecimento de padrões — e analisando sua relação com o desenvolvimento do raciocínio lógico e da resolução de problemas.

A seção 2.3 tratará das metodologias ágeis, apresentando seus valores, princípios e práticas, bem como discutindo a forma como esses conceitos vêm sendo adaptados em contextos educacionais, aproximando o aprendizado de abordagens típicas de ambientes corporativos e de desenvolvimento de software.

Por fim, na seção 2.4, serão abordados aspectos relacionados à qualidade de software, com ênfase em metodologias de teste e em práticas que favorecem a escrita de código mais confiável. Essa discussão visa conectar a importância do controle de qualidade no desenvolvimento de sistemas com o processo formativo de estudantes que estão iniciando no campo da programação.

2.1 Informática na Educação

Atualmente, os computadores e seus recursos estão amplamente presentes no contexto educacional, tornando-se instrumentos cada vez mais comuns e fundamentais para o processo de ensino-aprendizagem. A utilização dessas tecnologias abarca diversas técnicas pedagógicas que tornam as aulas mais dinâmicas, interativas e motivadoras para os alunos, contribuindo para um ambiente de aprendizagem envolvente.

A informática configura-se como uma nova abordagem para o desenvolvimento do conhecimento, proporcionando maior autonomia, independência e protagonismo por parte dos estudantes. Conforme Coutinho e Lisboa (2011), os jovens demandam modelos de aprendizagem que promovam a interação, estimulem a proatividade, incentivem a inovação e desenvolvam a

criatividade, ampliando seus horizontes de atuação.

De forma alinhada, o Ministério da Educação (MEC) reforça, no guia oficial de Educação Digital e Midiática, que a educação digital escolar e o pensamento computacional são elementos essenciais para o pleno exercício da cidadania digital. Destaca-se que o currículo estudantil deve contemplar três eixos estruturantes: o pensamento computacional, o mundo digital e a cultura digital. O eixo do pensamento computacional implica o desenvolvimento da habilidade de compreender, modelar e resolver problemas de maneira sistemática, criando e adaptando algoritmos e aplicando fundamentos da computação para o aprendizado criativo e crítico em múltiplas áreas, ponto que será mais explorado adiante. Já o eixo mundo digital engloba aprendizagens sobre artefatos digitais, proteção de dados e compreensão sobre hardware, software e a distribuição de informações; enquanto o eixo cultura digital promove o uso consciente, democrático e ético das tecnologias, desenvolvendo atitudes críticas e participação cidadã nas dinâmicas sociais mediadas pelas tecnologias digitais (Brasil, 2025)

Atualmente, o ensino de programação tem evoluído para além da transmissão de conceitos teóricos, incorporando metodologias que estimulam a participação ativa dos estudantes na construção do conhecimento. Nesse contexto, estratégias de aprendizagem ativa têm sido amplamente utilizadas para incentivar a resolução de problemas, a experimentação e a reflexão contínua sobre as soluções desenvolvidas. Diferentemente de abordagens tradicionais, centradas na exposição de conteúdo, essas metodologias promovem maior protagonismo do estudante, permitindo que ele valide hipóteses, identifique erros e aperfeiçoe suas soluções ao longo do processo de aprendizagem.

Entre os recursos que favorecem esse modelo de ensino, destacam-se os mecanismos de *feedback* automatizado, amplamente empregados em plataformas de ensino de programação e juízes *online*. Essas ferramentas permitem que o estudante obtenha retorno imediato sobre a corretude de sua solução, favorecendo ciclos contínuos de tentativa, análise e correção. Embora, na maioria desses ambientes, os testes sejam executados apenas após a implementação do código, esse modelo evidencia a importância da validação automática como instrumento de aprendizagem. O TDD amplia esse conceito ao propor que os testes sejam elaborados antes da implementação da solução, incentivando o estudante a compreender os requisitos do problema, definir o comportamento esperado do sistema e desenvolver o código de forma incremental, orientada por validações contínuas. Dessa forma, o TDD pode ser compreendido não apenas como uma prática de qualidade de software, mas também como uma estratégia pedagógica capaz

de fortalecer o raciocínio lógico, a autonomia e o pensamento computacional durante o ensino de programação introdutória.

Dessa forma, observa-se que a informática na educação vai além da simples utilização de recursos tecnológicos em sala de aula, abrangendo metodologias que promovem a participação ativa dos estudantes, o uso de *feedback* contínuo e a construção progressiva do conhecimento. Nesse contexto, práticas como o TDD apresentam potencial para enriquecer o ensino de programação introdutória ao incentivar a validação constante das soluções, a reflexão sobre os requisitos e o desenvolvimento estruturado de algoritmos. No próximo tópico, será abordado o conceito de pensamento computacional, destacando sua relação com essas estratégias pedagógicas e sua importância para o desenvolvimento das competências necessárias à formação de programadores.

2.2 Pensamento Computacional

O pensamento computacional é reconhecido como uma competência essencial para o século XXI e para a formação dos estudantes na era digital, indo além da programação ou uso restrito de computadores. Ele consolida-se como uma estratégia transversal de resolução de problemas complexos, com aplicações relevantes em áreas como biologia, medicina, economia, engenharia, ciências sociais e muito mais. Essa competência envolve habilidades como decomposição de grandes desafios em etapas menores, identificação de padrões, abstração para selecionar informações relevantes e elaboração de algoritmos, permitindo abordar questões contemporâneas de maneira sistemática, eficiente e criativa (Azevedo; Silva, 2025).

No contexto educacional, pesquisas apontam que a integração do pensamento computacional aos currículos escolares facilita o desenvolvimento do raciocínio lógico e analítico, estimula a criatividade, o pensamento crítico e prepara os estudantes para as demandas do futuro mercado de trabalho, cada vez mais impactado pela tecnologia. Segundo Brackmann (2023), trata-se de uma capacidade criativa, crítica e estratégica de utilizar os fundamentos da Ciência da Computação para solucionar problemas, tanto individualmente quanto em equipe.

O ensino do pensamento computacional agrega valor à formação integral do estudante, promovendo habilidades analíticas e tomadas de decisão assertivas. Organizações internacionais e documentos normativos como os CSTA K-12 Computer Science Standards (COMPUTER SCIENCE TEACHERS ASSOCIATION, 2011) e a BNCC Computação brasileira destacam a importância de desenvolver tais competências desde a Educação Infantil, de modo progressivo e interdisciplinar, favorecendo o desenvolvimento da autonomia intelectual e da capacidade de

resolver problemas em diferentes contextos.

Ademais, a aplicação do pensamento computacional na educação não está limitada ao uso de equipamentos digitais. Atividades desplugadas — como jogos, desafios lógicos e dinâmicas de grupo — também estimulam o raciocínio lógico, a colaboração e o uso de estratégias computacionais, contribuindo para a preparação dos estudantes frente às situações reais de maneira inovadora (Brasil, 2025).

Diante disso, a literatura e as diretrizes educacionais reforçam que o pensamento computacional é uma competência transversal e interdisciplinar, relevante para múltiplas áreas do conhecimento e formadora de uma base sólida para o aprendizado contínuo, o desenvolvimento profissional e a cidadania ativa na sociedade contemporânea.

Dando continuidade à abordagem de práticas inovadoras no contexto educacional, o próximo tópico será dedicado à Metodologia Ágil. Serão apresentados seus conceitos, benefícios e estratégias de aplicação na educação, destacando como essa abordagem, originária do setor tecnológico, pode contribuir para um ambiente escolar mais dinâmico, colaborativo e flexível, favorecendo o protagonismo dos alunos e a adaptação constante às necessidades e desafios do processo de ensino-aprendizagem.

2.3 Metodologia Ágil

Construir projetos em torno de indivíduos motivados é um dos 12 princípios centrais do Manifesto para o Desenvolvimento Ágil, criado por um grupo de desenvolvedores em 2001. Esse manifesto marcou a primeira discussão formal sobre metodologias ágeis, inicialmente focadas em otimizar processos de desenvolvimento de software. No entanto, suas práticas têm sido amplamente aplicadas a diversas áreas, incluindo a educação, devido à sua ênfase em flexibilidade, colaboração e entrega contínua de valor.

Ao incorporar a metodologia ágil na educação, o processo de ensino se torna mais flexível e dinâmico, permitindo que a aprendizagem flua mesmo dentro de cronogramas estruturados. Essa abordagem promove a divisão do ensino em etapas, nas quais avaliações periódicas são usadas para identificar e potencializar o desempenho individual dos estudantes.

Os benefícios das metodologias ágeis na educação incluem o incentivo à conscientização dos alunos sobre seus próprios processos de aprendizagem, enquanto os professores atuam como facilitadores, observando e intervindo conforme necessário. A transparência inerente a essas metodologias possibilita um acompanhamento mais eficiente do progresso e do desenvolvimento

dos estudantes, contribuindo para uma educação mais personalizada e eficaz.

Além disso, um aspecto complementar à qualidade da aprendizagem que será abordado em seguida refere-se à qualidade de software, destacando a importância de metodologias de teste e garantia da excelência nos processos educacionais e tecnológicos.

Como continuidade da discussão sobre abordagens inovadoras e metodologias aplicadas no contexto educacional, o próximo tópico abordará o tema da qualidade de software. Será explorada a importância dos processos e critérios técnicos para garantir a confiabilidade, usabilidade e eficiência dos softwares educacionais, essenciais para apoiar o ensino-aprendizagem de maneira eficaz e alinhada às necessidades dos estudantes e educadores.

2.4 Qualidade de Software

A qualidade de software envolve práticas e princípios que asseguram a funcionalidade, eficiência, segurança e manutenibilidade do código ao longo do tempo. No ensino de programação, é fundamental que os alunos compreendam o papel dos testes, incluindo testes unitários, TDD e testes de integração. O TDD, em particular, contribui para a melhoria da qualidade do código ao incentivar o programador a validar o software antes mesmo de sua implementação. A incorporação de testes desde as fases iniciais do aprendizado prepara os estudantes para desenvolver códigos mais robustos e confiáveis. Como salientado por Martins em 2019, práticas como código limpo e testes favorecem um processo de desenvolvimento sustentável, formando profissionais mais qualificados. Estudos também indicam que a criação precoce de casos de teste estimula o pensamento computacional e promove um raciocínio estruturado por meio de ciclos iterativos de tentativa e erro.

Os testes de software constituem um dos principais mecanismos para verificação e validação de sistemas, permitindo identificar defeitos, verificar se os requisitos foram atendidos e aumentar a confiabilidade das aplicações desenvolvidas (Pressman; Maxim, 2016; Sommerville, 2011). Sua utilização contribui para reduzir falhas durante o ciclo de desenvolvimento, facilitar a manutenção do código e apoiar a evolução contínua do software (Myers; Sandler; Badgett, 2011). De forma geral, os testes podem ser classificados em diferentes categorias, de acordo com seu objetivo e nível de abrangência, destacando-se os testes unitários, de integração, de sistema e de aceitação (**??**). Enquanto os testes unitários verificam o comportamento de componentes individuais, os testes de integração avaliam a interação entre diferentes módulos, os testes de sistema validam o funcionamento da aplicação como um todo e os testes de aceitação verificam

se o software atende às necessidades do usuário ou do cliente (Pressman; Maxim, 2016).

Outra classificação amplamente utilizada diferencia os testes em caixa branca (*white-box*) e caixa preta (*black-box*) (Ammann; Offutt, 2016). Os testes de caixa branca consideram a estrutura interna do código-fonte, permitindo avaliar caminhos de execução, estruturas de decisão e cobertura do código (Myers; Sandler; Badgett, 2011). Já os testes de caixa preta concentram-se na validação das funcionalidades do sistema a partir das entradas e saídas esperadas, sem considerar sua implementação interna (Sommerville, 2011). Em ambientes educacionais, ambos os tipos podem ser explorados de maneira complementar, favorecendo tanto a compreensão da lógica implementada quanto a verificação dos requisitos funcionais das soluções desenvolvidas pelos estudantes.

Além dos diferentes tipos de testes, a literatura apresenta critérios que auxiliam na avaliação da qualidade das suítes de teste, como cobertura de instruções, cobertura de decisões, cobertura de caminhos e análise de casos de entrada e saída (Ammann; Offutt, 2016). Esses critérios permitem verificar o nível de abrangência dos testes elaborados e identificar cenários ainda não contemplados durante a validação do software. No contexto do ensino de programação, tais critérios também podem ser utilizados como instrumentos pedagógicos para incentivar os estudantes a refletirem sobre diferentes possibilidades de execução e sobre o comportamento esperado de suas soluções.

Nos últimos anos, plataformas de correção automática e juízes *online* passaram a incorporar mecanismos de execução automática de testes e geração de *feedback* imediato aos estudantes (Ihantola *et al.*, 2010). Nesses ambientes, os programas submetidos são avaliados por conjuntos de casos de teste previamente definidos, permitindo identificar rapidamente erros de implementação, inconsistências nos algoritmos e falhas relacionadas aos requisitos propostos (Douce; Livingstone; Orwell, 2005). O retorno automático fornecido por essas ferramentas favorece ciclos contínuos de aprendizagem, nos quais o estudante implementa, valida e aperfeiçoa sua solução de forma iterativa. Essa característica aproxima o ensino de programação de práticas modernas de desenvolvimento de software, nas quais a validação contínua constitui um elemento fundamental do processo de construção de sistemas.

Nesse contexto, o TDD destaca-se como uma prática orientada à qualidade de software que propõe a criação de testes antes da implementação da funcionalidade. Popularizada por Beck no contexto do desenvolvimento ágil, essa metodologia busca transformar os testes em um elemento ativo do processo de construção do software, e não apenas em uma etapa posterior de

validação (Beck, 2003). O princípio central do TDD consiste em definir previamente o comportamento esperado do sistema por meio de testes automatizados, permitindo que o desenvolvimento seja guiado pelos requisitos e pelos resultados obtidos durante a execução desses testes.

A aplicação do TDD ocorre tradicionalmente por meio de ciclos curtos conhecidos como *Red-Green-Refactor*. Na etapa *Red*, é elaborado um teste que inicialmente falha, representando uma funcionalidade ainda não implementada. Em seguida, na etapa *Green*, desenvolve-se o código mínimo necessário para que o teste seja aprovado. Por fim, na etapa *Refactor*, realiza-se a melhoria da estrutura interna do código, eliminando redundâncias e aumentando sua legibilidade sem alterar seu comportamento externo. Esse processo iterativo favorece a construção incremental do software e incentiva a realização constante de validações durante o desenvolvimento (Martin, 2019).

Além dos benefícios técnicos relacionados à redução de defeitos e ao aumento da manutenibilidade, o TDD também apresenta potencial formativo quando aplicado ao ensino de programação. Ao exigir que o estudante reflita previamente sobre entradas, saídas e comportamentos esperados antes da implementação, a abordagem estimula habilidades associadas ao pensamento computacional, como abstração, decomposição de problemas e construção lógica das soluções. Vale destacar que, no ensino de programação, já existem ferramentas como juízes online e ambientes de correção automática que utilizam casos de entrada e saída previamente definidos, incluindo conjuntos de testes ocultos, para avaliar as soluções submetidas pelos estudantes. Nesses sistemas, o código é desenvolvido livremente e validado apenas ao final, quando a execução dos testes indica aprovação ou falha. Embora essas ferramentas sejam importantes para a verificação automática de exercícios, sua lógica difere da proposta do TDD, pois a avaliação ocorre de forma posterior ao desenvolvimento. Já no TDD, os testes são elaborados antes da implementação e passam a orientar a construção incremental da solução, promovendo uma relação mais ativa entre teste, código e aprendizado. Dessa forma, o TDD pode ser compreendido como uma abordagem complementar às práticas já presentes em ambientes educacionais, ampliando o papel dos testes de um mecanismo de correção para um instrumento de formação e reflexão sobre o processo de desenvolvimento.

Nesse contexto, Feitosa, Silva e Fabri (2021) propuseram uma abordagem que integra o TDD com critérios escalonados de testes, alinhados a objetivos específicos de aprendizagem. Essa metodologia associa cada critério a conceitos de programação, permitindo o desenvolvimento progressivo das habilidades dos estudantes. Pesquisas recentes, como a de Araujo, Flach e

Bittencourt (2024), evidenciam que o uso de testes unitários em cursos introdutórios de *Python* resulta em taxas de sucesso superiores a 90% nos projetos dos alunos.

Além disso, fornecer mensagens de erro claras e contextualizadas na língua nativa dos estudantes tem se mostrado eficaz para reduzir a frustração e facilitar a assimilação dos conceitos. Combinadas, essas práticas fortalecem não apenas a qualidade do código produzido, mas também a autonomia e a confiança dos alunos durante sua trajetória de aprendizagem. Destaca-se ainda a relevância dos *feedbacks* imediatos e da avaliação baseada em casos de teste para reforçar o aprendizado, identificar e corrigir erros rapidamente, acelerando o progresso e aumentando a confiança dos estudantes.

Por fim, os conceitos apresentados neste capítulo evidenciam que a integração entre informática na educação, pensamento computacional, metodologias ágeis e práticas de qualidade de software constitui uma base relevante para a modernização do ensino de programação introdutória. A literatura demonstra que estratégias voltadas à resolução estruturada de problemas, ao desenvolvimento incremental e à validação contínua do código podem contribuir tanto para o aprimoramento técnico quanto para o fortalecimento das habilidades cognitivas dos estudantes. Nesse contexto, o uso de práticas como testes automatizados e TDD apresenta-se como uma alternativa promissora para aproximar o ambiente acadêmico das demandas contemporâneas da indústria de software, favorecendo uma aprendizagem mais ativa, reflexiva e alinhada às competências exigidas no mercado de tecnologia. Esses fundamentos servem como suporte teórico para a investigação proposta neste trabalho e orientam a análise apresentada nos capítulos seguintes.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados trabalhos e estudos relacionados às abordagens baseadas em testes para o ensino de programação introdutória, que exploram diferentes metodologias e seus impactos no processo educativo. Estas pesquisas discutem a importância da integração dos testes desde os primeiros passos do aluno na formação em programação, destacando os benefícios dessa prática para a melhoria da qualidade do ensino e para os resultados obtidos pelos estudantes ao adotar essa abordagem.

3.1 Test-Driven Development: Challenges and Recommendations for Trainers and Educators

Trabalhos como o de Paez et al. (2024), constituem uma contribuição relevante para as pesquisas sobre ensino de programação com enfoque em práticas baseadas em testes. Os autores conduzem uma análise temática a partir de entrevistas com profissionais e educadores experientes de diversos países, buscando identificar os principais desafios e recomendações associados ao ensino de TDD.

A investigação aponta seis desafios centrais e vinte e uma recomendações organizadas em eixos temáticos, tais como compreensão conceitual do TDD, limitações tecnológicas, prática estruturada e aspectos pedagógicos. Entre os desafios mais recorrentes destacam-se: a dificuldade de promover a mudança de mentalidade necessária para adoção do TDD; a insuficiência de conhecimentos prévios por parte dos estudantes; e a presença de barreiras técnicas que se intensificam em ambientes e projetos mais complexos.

No que se refere às recomendações, o estudo ressalta a importância de incentivar a prática contínua e sistemática, fortalecer o protagonismo do educador enquanto praticante de TDD, integrar teoria e demonstrações práticas, distribuir o ensino dessa metodologia ao longo de diferentes disciplinas e utilizar estratégias colaborativas, como programação em pares.

Assim, os resultados apresentados por Paez et al. (2024) oferecem subsídios significativos para a formulação de metodologias pedagógicas mais eficazes, tanto no contexto da educação formal quanto em programas de formação profissional, reforçando a relevância de abordagens estruturadas para o ensino de TDD.

3.2 Qual tipo de feedback ajuda os estudantes novatos em programação? Um estudo exploratório com múltiplos feedbacks em um curso introdutório de Python

No artigo desenvolvido por Araujo, Flach e Bittencourt (2024), apresenta-se um estudo de caso que investiga o uso e a eficácia de diferentes tipos de *feedback* no contexto do ensino introdutório de programação. A pesquisa foi realizada em um curso de quatro semanas utilizando o ambiente PEEF, que disponibiliza aos estudantes diversos mecanismos de apoio, tais como testes unitários, mensagens de erro aprimoradas e interação por meio de chatbot.

Os resultados evidenciam que os testes unitários foram o recurso mais amplamente utilizado, presentes em 83,6% dos projetos desenvolvidos e com 90,5% de sucesso nos testes aplicados. As mensagens de erro melhoradas, adaptadas para o português, mostraram-se úteis para reduzir a frustração e favorecer a compreensão das falhas, embora tenham apresentado limitações quanto à precisão das orientações. Já o chatbot teve baixa adesão e utilidade, sendo explorado por poucos estudantes ao longo do curso.

O estudo destaca ainda a relevância do *feedback* imediato para o processo de aprendizagem, observando-se aumento da confiança dos estudantes e uma melhora significativa na percepção quanto ao uso de testes e mensagens de erro após participação no curso. Os autores ressaltam a necessidade de avanços na personalização dos *feedbacks* e de estratégias que incentivem maior engajamento com os recursos digitais de suporte.

Assim, o trabalho contribui para o entendimento dos benefícios e desafios associados à integração de múltiplos tipos de *feedback* no ensino inicial de programação, oferecendo subsídios importantes para o aprimoramento de práticas pedagógicas no âmbito da computação educacional.

3.3 Desenvolvimento baseado em testes com aplicação de critérios de teste no ensino introdutório de computação

O estudo de Feitosa et al. (2021) apresenta uma proposta de metodologia que integra TDD a critérios escalonados de testes de software vinculados a objetivos específicos de aprendizagem em disciplinas introdutórias de programação. A abordagem busca aprimorar o desenvolvimento do raciocínio computacional dos estudantes, incentivando-os a estruturar o pensamento e planejar soluções antes da implementação, reduzindo práticas baseadas exclusivamente em tentativa e erro.

O estudo enfatiza o papel da criação e automação de casos de teste como elementos centrais do processo de aprendizagem, proporcionando avaliação contínua e *feedback* imediato. Essa estratégia promove maior consciência sobre a qualidade do código produzido e favorece a autonomia do estudante, que passa a compreender o teste não apenas como mecanismo de verificação, mas como instrumento formativo.

Os autores argumentam que a adoção de testes desde os primeiros contatos com a programação contribui para a formação de programadores mais críticos, cuidadosos e alinhados com práticas contemporâneas de desenvolvimento de software. Nesse sentido, o trabalho constitui uma referência relevante para investigações que analisam o impacto da integração de testes no ensino de programação, oferecendo fundamentos importantes para o aprimoramento de práticas pedagógicas na área.

3.3.1 Síntese Comparativa dos Trabalhos Relacionados

A Tabela 1 apresenta uma síntese dos principais estudos analisados nesta pesquisa, destacando seus objetivos, métodos empregados, participantes, principais resultados, limitações e a relação de cada trabalho com a presente investigação. Essa comparação permite identificar convergências entre as pesquisas e evidenciar a contribuição deste trabalho em relação aos estudos existentes.

3.3.2 Síntese Comparativa dos Trabalhos Relacionados

A Tabela 1 apresenta uma síntese dos principais estudos analisados nesta pesquisa, destacando seus objetivos, métodos empregados, participantes, principais resultados, limitações e a relação de cada trabalho com a presente investigação. Essa comparação permite identificar convergências entre as pesquisas e evidenciar a contribuição deste trabalho em relação aos estudos existentes.

3.3.3 Síntese Comparativa dos Trabalhos Relacionados

A Tabela 1 apresenta uma síntese dos principais estudos analisados nesta pesquisa, destacando seus objetivos, métodos empregados, participantes, principais resultados, limitações e a relação de cada trabalho com a presente investigação. Essa comparação permite identificar convergências entre as pesquisas e evidenciar a contribuição deste trabalho em relação aos

estudos existentes.

Tabela 1 – Comparação entre os trabalhos relacionados

Autor/Ano	**Objetivo**	**Método**	**Participantes**	**Principais resultados**	**Limitações**	**Relação com este TCC**
Paez et al. (2024)	Identificar desafios e recomendações para o ensino de TDD.	Entrevistas	Educadores e profissionais	Seis desafios e 21 recomendações para adoção do TDD.	Não houve aplicação prática em disciplinas.	Fundamenta os desafios pedagógicos e orienta a proposta deste trabalho.
Araujo et al. (2024)	Avaliar diferentes tipos de feedback no ensino de programação.	Estudo de caso	Estudantes de Python	Testes unitários e feedback imediato melhoraram a aprendizagem.	Aplicação restrita a um único curso.	Reforça o uso de feedback automatizado como apoio ao ensino.
Feitosa et al. (2021)	Integrar TDD e critérios de teste ao ensino.	Proposta metodológica	Disciplinas introdutórias	Relaciona testes aos objetivos de aprendizagem.	Sem validação longitudinal.	Base principal para a proposta de implantação apresentada neste trabalho.

Fonte: Autoria Própria (2026).

A análise comparativa evidencia que os três trabalhos convergem ao reconhecer o potencial das práticas baseadas em testes para apoiar o ensino de programação introdutória, embora abordem o tema sob perspectivas distintas. Enquanto Paez, Fontdevila e Zangara (2024) concentram-se nos desafios pedagógicos e nas recomendações para a adoção do TDD, Araujo, Flach e Bittencourt (2024) investigam o papel do *feedback* automatizado como mecanismo de apoio à aprendizagem, e Feitosa et al. (2021) apresentam uma proposta metodológica voltada à integração entre TDD, critérios de teste e objetivos de aprendizagem.

Observa-se ainda que nenhum dos estudos reúne simultaneamente uma revisão da literatura, uma investigação sobre a percepção de estudantes, docentes e profissionais e a proposição de um fluxo estruturado para implantação do TDD no ensino introdutório de programação. Nesse contexto, este trabalho diferencia-se por integrar essas três perspectivas. Inicialmente, realiza um levantamento bibliográfico para identificar desafios e oportunidades relacionados ao uso do TDD; posteriormente, conduz um *survey* para compreender a percepção de diferentes perfis de participantes sobre testes de software e TDD; por fim, propõe uma abordagem gradual de implantação da metodologia, fundamentada tanto nas evidências da literatura quanto nos resultados obtidos na pesquisa. Dessa forma, busca-se contribuir para a aproximação entre as práticas de qualidade de software adotadas pela indústria e o processo de ensino-aprendizagem da programação introdutória.

4 METODOLOGIA

Este trabalho caracteriza-se como uma pesquisa de natureza descritiva, com abordagem predominantemente quantitativa, buscando analisar a aplicabilidade do TDD no ensino introdutório de programação a partir de evidências identificadas na literatura e de dados coletados por meio de uma pesquisa do tipo *survey*. A metodologia foi estruturada em etapas complementares, contemplando o levantamento bibliográfico, a coleta de percepções de estudantes, docentes e profissionais da área de tecnologia e a análise dos dados obtidos, de modo a garantir rigor científico, clareza analítica e aderência aos objetivos da investigação. A seguir, cada etapa é detalhada em seções específicas.

4.1 Levantamento Bibliográfico

A primeira etapa da metodologia consistiu na realização de um levantamento bibliográfico exploratório, com foco em estudos relacionados ao uso de testes de software e à aplicação do TDD no ensino introdutório de programação. O objetivo dessa etapa foi reunir e analisar evidências teóricas e empíricas disponíveis na literatura, buscando compreender benefícios, desafios e possibilidades de aplicação dessa abordagem no contexto educacional.

A busca foi realizada nas bases e repositórios acadêmicos *IEEE Xplore*, *ACM Digital Library*, *ResearchGate*, Sociedade Brasileira de Computação (SBC), repositório institucional da UFERSA e na plataforma *LitMaps*, utilizada como apoio para mapeamento de referências e identificação de trabalhos relacionados. O período considerado contemplou publicações dos últimos vinte anos, entre 2005 e 2025, com prioridade para estudos mais recentes sobre TDD, testes automatizados, *feedback* automático e ensino de programação introdutória.

Foram utilizadas combinações de palavras-chave em português e inglês, tais como: *Test-Driven Development*, TDD, ensino de programação, programação introdutória, testes automatizados, testes unitários, *feedback* automático, juízes online, *introductory programming*, *software testing education*, *automated feedback* e *computational thinking*. As buscas consideraram a ocorrência desses termos em títulos, resumos e palavras-chave dos trabalhos.

Como critérios de inclusão, foram considerados estudos que: abordassem o uso de TDD ou testes de software no ensino de programação; discutissem práticas de *feedback* automatizado ou avaliação baseada em testes; apresentassem relação com disciplinas introdutórias de computação; ou contribuíssem teoricamente para a compreensão de qualidade de software, pensamento

computacional e metodologias de ensino aplicadas à programação. Foram excluídos trabalhos sem relação direta com o ensino de programação, estudos duplicados, materiais sem acesso ao texto completo e publicações com foco exclusivamente industrial, sem contribuição para o contexto educacional.

Inicialmente, foram identificados estudos nas bases consultadas. Após a remoção de duplicidades e a leitura dos títulos e resumos, parte desses trabalhos foi pré-selecionada. Em seguida, após a leitura completa e aplicação dos critérios de inclusão e exclusão, um conjunto de estudos foi selecionado para análise detalhada e utilizado como base para a fundamentação teórica, discussão dos resultados e elaboração da proposta desta pesquisa.

A análise dos estudos selecionados permitiu identificar aspectos recorrentes, como: dificuldades na inserção do TDD em disciplinas introdutórias; uso de ferramentas e mecanismos de *feedback* automatizado como suporte ao aprendizado; comparação entre abordagens tradicionais e práticas baseadas em testes; impactos dos testes automatizados no desenvolvimento do raciocínio lógico e do pensamento computacional; e influência dos testes na melhoria da qualidade do código e na autonomia dos estudantes.

Dessa forma, o levantamento bibliográfico não apenas fundamentou o referencial teórico do trabalho, como também orientou a elaboração do instrumento de coleta de dados e a definição das dimensões analisadas no *survey*. A partir das evidências identificadas, foi possível construir uma base analítica para compreender como práticas baseadas em testes podem contribuir para o ensino de programação introdutória e para a aproximação entre a formação acadêmica e as práticas contemporâneas da indústria de software.

4.2 Survey

Em complemento ao levantamento bibliográfico, a pesquisa adotou a metodologia do tipo *survey* como estratégia para obtenção de evidências empíricas relacionadas ao tema investigado. Essa abordagem foi escolhida por possibilitar a coleta estruturada de dados junto a estudantes, docentes e profissionais da área de tecnologia, permitindo identificar percepções, experiências e níveis de familiaridade com práticas baseadas em testes, especialmente o *Test-Driven Development* (TDD).

A aplicação do *survey* teve como objetivo ampliar a compreensão sobre como o uso de testes de software e do TDD é percebido tanto no contexto educacional quanto no ambiente profissional, possibilitando observar tendências, padrões de adoção e fatores associados às

dificuldades e oportunidades de utilização dessa metodologia. Para isso, foi utilizado um questionário estruturado, elaborado a partir dos aspectos identificados na revisão da literatura e alinhado aos objetivos da pesquisa.

Por se tratar de uma investigação com abordagem predominantemente quantitativa e caráter descritivo, os dados coletados permitiram analisar a distribuição das respostas, identificar comportamentos recorrentes e compreender como os participantes avaliam a contribuição do TDD para aspectos como qualidade do código, desenvolvimento do raciocínio lógico e aproximação com práticas utilizadas na indústria de software. Dessa forma, o *survey* complementa as evidências teóricas obtidas na literatura, contribuindo para uma análise mais ampla e fundamentada sobre a aplicabilidade do TDD no ensino introdutório de programação.

4.3 Coleta de dados

Para a coleta de dados, foram utilizados questionários estruturados direcionados a estudantes de cursos da área de tecnologia e computação, profissionais atuantes em Tecnologia da Informação e docentes vinculados a instituições de ensino brasileiras. O objetivo dessa etapa foi identificar o cenário atual relacionado ao contato com testes de software e à percepção sobre o uso do TDD no contexto do ensino introdutório de programação. O instrumento de coleta permitiu levantar informações sobre práticas adotadas, dificuldades enfrentadas, nível de familiaridade com metodologias baseadas em testes e percepção quanto à aplicabilidade dessa abordagem no processo de aprendizagem e no contexto profissional.

A aplicação do questionário ocorreu de forma *online*, permitindo alcançar participantes com diferentes perfis e níveis de experiência na área de tecnologia. Ao todo, participaram da pesquisa 32 respondentes, sendo 16 estudantes, 14 profissionais da área de Tecnologia da Informação e 2 docentes de instituições de ensino. Essa composição possibilitou reunir percepções provenientes tanto do contexto acadêmico quanto do ambiente profissional, enriquecendo a análise sobre o ensino de programação e o uso de testes de software. Os dados coletados permitiram compreender, de maneira estruturada, aspectos relacionados ao uso de testes de software, aos desafios para adoção do TDD e às percepções sobre seus possíveis benefícios para o desenvolvimento do raciocínio lógico, a organização das soluções e a qualidade do código produzido.

Essa etapa constituiu um elemento central da investigação, pois permitiu relacionar os achados identificados na revisão da literatura com percepções oriundas de contextos reais

de formação e atuação profissional. Dessa forma, os resultados obtidos contribuíram para a construção de um diagnóstico mais consistente sobre o uso de práticas baseadas em testes e forneceram subsídios para a elaboração da proposta de abordagem gradual do TDD no ensino introdutório de programação.

O Apêndice A apresenta, além do questionário utilizado na pesquisa, uma tabela de rastreabilidade que relaciona cada questão aos objetivos específicos deste trabalho. Esse mapeamento evidencia como o instrumento de coleta foi concebido para atender aos objetivos da investigação e justifica a utilização de cada pergunta na análise dos resultados.

4.4 Análise de dados

Por fim, a análise dos dados foi conduzida por meio de técnicas quantitativas de caráter descritivo, com o objetivo de organizar, sintetizar e interpretar as informações coletadas durante a aplicação do *survey*. Essa abordagem permitiu examinar as respostas obtidas de forma estruturada, identificando padrões, frequências e tendências relacionadas ao contato com testes de software, ao nível de familiaridade com o TDD e às percepções sobre sua aplicabilidade no ensino introdutório de programação.

Para o tratamento dos dados, foram consideradas medidas descritivas e análises de distribuição das respostas obtidas no questionário, permitindo observar comportamentos recorrentes entre estudantes, docentes e profissionais da área de tecnologia. As questões objetivas e em escala possibilitaram identificar relações entre experiência prévia com testes, percepção sobre benefícios do TDD e sua associação com aspectos como compreensão de requisitos, organização das soluções, qualidade do código e desenvolvimento do raciocínio lógico.

A análise dos resultados também permitiu estabelecer conexões entre os dados empíricos e os achados identificados na revisão da literatura, contribuindo para verificar em que medida as percepções dos participantes convergem com os desafios e benefícios apontados por estudos anteriores. Dessa forma, esta etapa forneceu subsídios para responder à questão central da pesquisa e apoiar a construção da proposta de abordagem gradual para inserção do TDD no ensino introdutório de programação.

Essa etapa foi essencial para consolidar evidências sobre a aplicabilidade do TDD no contexto educacional investigado, permitindo compreender oportunidades e limitações relacionadas à sua adoção e contribuindo para uma discussão mais fundamentada sobre práticas de qualidade de software aplicadas ao processo de ensino-aprendizagem.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos a partir das etapas metodológicas desenvolvidas ao longo da pesquisa, com foco na análise da aplicabilidade do TDD no ensino introdutório de programação. Inicialmente, são apresentados os achados do levantamento bibliográfico, realizado com o objetivo de identificar evidências científicas sobre o uso de testes de software e metodologias baseadas em testes no contexto educacional, destacando benefícios, limitações e desafios relacionados à adoção do TDD.

Em seguida, são discutidos os elementos identificados na literatura que subsidiaram a elaboração do instrumento de coleta de dados e orientaram as dimensões investigadas na pesquisa do tipo *survey*. O questionário foi aplicado a estudantes, docentes e profissionais da área de tecnologia, buscando compreender experiências, percepções e níveis de familiaridade relacionados ao uso de testes de software e à metodologia TDD.

Posteriormente, são apresentados e analisados os dados obtidos por meio do questionário, utilizando técnicas quantitativas de caráter descritivo para identificar padrões, tendências e relações entre as respostas coletadas. A análise buscou compreender fatores que influenciam a adoção do TDD, sua percepção quanto à contribuição para o raciocínio lógico, qualidade do código e aproximação entre o ensino introdutório e as práticas da indústria de software.

Por fim, o capítulo apresenta a discussão dos resultados e a proposta de uma abordagem gradual para inserção do TDD no ensino introdutório de programação, construída a partir da integração entre os achados da literatura e as evidências empíricas coletadas. O objetivo dessa proposta é fornecer subsídios para apoiar futuras iniciativas pedagógicas voltadas à introdução progressiva de práticas de qualidade de software no processo de ensino-aprendizagem.

5.1 Levantamento Bibliográfico

Durante o levantamento bibliográfico, utilizou-se a plataforma *LitMaps*[1] como ferramenta de apoio para o mapeamento, rastreamento e organização da produção científica relacionada ao tema desta pesquisa. A utilização da plataforma permitiu identificar estudos relevantes sobre a aplicação do TDD no ensino introdutório de programação, além de possibilitar a visualização das relações entre autores, citações e linhas de investigação desenvolvidas ao longo do tempo. Esse processo contribuiu para a seleção sistemática dos trabalhos que serviram como base para

[1] Disponível em: https://www.litmaps.com/

a construção do referencial teórico e para a definição dos elementos investigados nas etapas posteriores do estudo.

Para a busca, consulta e análise dos materiais científicos, foram utilizadas bases e repositórios acadêmicos reconhecidos na área de Computação, como *Digital Library*[2], *IEEE Xplore*[3], *ResearchGate*[4], Sociedade Brasileira de Computação (SBC)[5] e o repositório institucional da UFERSA[6]. A consulta a essas plataformas possibilitou reunir estudos publicados ao longo dos últimos vinte anos, proporcionando uma visão abrangente do estado da arte sobre qualidade de software, metodologias ágeis, testes automatizados e abordagens baseadas em testes aplicadas ao contexto educacional.

O levantamento bibliográfico teve como objetivo compreender como práticas de teste de software vêm sendo incorporadas ao ensino de programação introdutória, com ênfase em estudos que investigam o uso do TDD, testes automatizados e mecanismos de *feedback* como instrumentos de apoio ao processo de ensino-aprendizagem. Além da identificação de benefícios pedagógicos, buscou-se analisar limitações, desafios de implementação e fatores que influenciam a adoção dessas abordagens em ambientes educacionais.

A partir da análise dos trabalhos selecionados, foram identificados aspectos recorrentes e relevantes para o desenvolvimento desta pesquisa, dentre os quais destacam-se:

- Dificuldades relacionadas à inserção e consolidação do TDD em disciplinas introdutórias de programação;
- Uso de ferramentas e mecanismos de *feedback* automatizado como suporte ao processo de aprendizagem;
- Comparações entre grupos de estudantes submetidos a abordagens tradicionais e grupos que utilizaram práticas baseadas em testes;
- Impactos do uso de testes automatizados no desenvolvimento do raciocínio lógico, da abstração e do pensamento computacional;
- Influência da utilização de testes na qualidade do código produzido e no fortalecimento da autonomia dos estudantes.

Os resultados do levantamento permitiram identificar tendências e lacunas ainda presentes na literatura, além de evidenciar oportunidades para ampliação do uso de práticas baseadas

[2] Disponível em: https://digitallibrary.io/
[3] Disponível em: https://ieeexplore.ieee.org/
[4] Disponível em: https://www.researchgate.net/
[5] Disponível em: https://www.sbc.org.br/
[6] Disponível em: https://repositorio.ufersa.edu.br/

em testes no contexto educacional. Observou-se que, embora existam desafios relacionados à adoção do TDD, como curva de aprendizagem inicial, adaptação metodológica e necessidade de capacitação, os estudos apontam benefícios associados à melhoria da qualidade das soluções desenvolvidas, ao fortalecimento do pensamento computacional e à aproximação entre a formação acadêmica e as práticas contemporâneas adotadas pela indústria de software. Esses achados serviram como base para a construção do instrumento de coleta de dados e para a proposta discutida nos capítulos seguintes.

5.2 Aplicação de Survey

Durante a revisão da literatura, estudos como o de Paez, Fontdevila e Zangara (2024) demonstram que a inclusão do *Test-Driven Development* (TDD) em contextos educacionais enfrenta desafios que vão além da simples apresentação dos conceitos técnicos. Entre os principais obstáculos identificados pelos autores destacam-se a necessidade de mudança de *mindset* para uma abordagem baseada em *test-first*, a insuficiência de conhecimentos prévios sobre testes de software e programação orientada a objetos, bem como a dificuldade de aplicar o TDD em cenários mais complexos, como sistemas legados, interfaces gráficas e aplicações com elevado número de efeitos colaterais. Os autores também ressaltam que a curva de aprendizagem da metodologia tende a ser gradual, exigindo tempo para que os estudantes desenvolvam autonomia na elaboração e organização de testes.

Essas evidências indicam que a adoção do TDD está fortemente relacionada às estratégias pedagógicas empregadas, ao nível de preparo dos estudantes e ao apoio oferecido pelos docentes, e não apenas ao domínio dos conceitos técnicos. Assim, compreender as percepções e experiências dos diferentes atores envolvidos no processo de ensino-aprendizagem torna-se fundamental para identificar fatores que favorecem ou dificultam a utilização dessa abordagem no contexto educacional.

Com base nessas evidências, este trabalho adotou a metodologia do tipo *survey* como estratégia de investigação, por meio da aplicação de um questionário estruturado a estudantes, docentes e profissionais da área de tecnologia. O instrumento foi elaborado a partir dos aspectos identificados durante a revisão da literatura, buscando coletar dados quantitativos sobre o nível de familiaridade com o TDD, o uso de testes de software, as dificuldades encontradas durante sua adoção e a percepção dos participantes quanto aos seus potenciais benefícios para o ensino introdutório de programação e para a aproximação entre a formação acadêmica e as práticas da

indústria de software. Na seção seguinte, são apresentados o instrumento de coleta de dados e os procedimentos adotados para sua aplicação.

5.2.1 Coleta de dados

A coleta de dados desta pesquisa foi realizada por meio de uma abordagem do tipo *survey*, utilizando como instrumento um questionário estruturado elaborado na plataforma *Google Forms*[7]. Essa estratégia foi adotada com o objetivo de coletar, de forma sistemática, informações junto a estudantes, docentes e profissionais da área de tecnologia, possibilitando a identificação de percepções, experiências e níveis de familiaridade relacionados ao uso de testes de software e da metodologia *Test-Driven Development* (TDD).

A elaboração do questionário foi fundamentada nos desafios, lacunas e recomendações identificados durante a revisão da literatura, especialmente nos estudos de Feitosa, Silva e Fabri (2021), Paez, Fontdevila e Zangara (2024) e Araujo, Flach e Bittencourt (2024), que abordam a utilização de testes de software e do TDD no contexto educacional. As questões foram construídas de modo a atender aos objetivos da pesquisa, contemplando aspectos relacionados ao perfil dos participantes, ao contato com testes de software, ao conhecimento sobre o TDD e às percepções quanto aos benefícios e desafios da utilização dessa abordagem no ensino introdutório de programação.

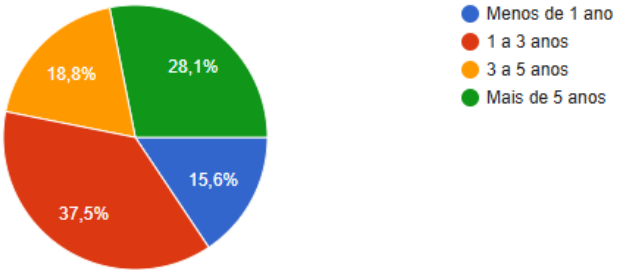
O instrumento de coleta foi composto por 14 questões, distribuídas entre perguntas objetivas de múltipla escolha, questões de seleção única e itens estruturados em escala de *Likert*. Essa organização possibilitou a obtenção de dados quantitativos, permitindo identificar padrões, frequências e tendências relacionados à utilização de testes de software e à percepção dos participantes sobre a aplicabilidade do TDD no contexto acadêmico e profissional.

A aplicação do questionário ocorreu de forma *online*, no período de 18 de março a 6 de abril de 2026, garantindo facilidade de acesso e maior alcance dos participantes. A divulgação foi realizada por meio de redes acadêmicas, grupos de tecnologia e contatos institucionais, buscando reunir uma amostra composta por estudantes, docentes e profissionais com diferentes níveis de experiência na área de Tecnologia da Informação.

Os dados obtidos por meio do *survey* constituíram a base empírica desta pesquisa, permitindo analisar o cenário atual de utilização de testes de software, o nível de conhecimento sobre o TDD e as percepções dos participantes quanto aos seus potenciais benefícios para o

[7] Disponível em: https://forms.gle/iktTrAxgkrbJDuoD8

Figura 1 – Tempo de experiência com programação.



Fonte: Autoria Própria (2026).

desenvolvimento do raciocínio lógico, da qualidade do código e da aproximação entre o ensino de programação introdutória e as práticas adotadas pela indústria de software.

5.2.2 Análise de Dados

A análise dos dados foi realizada a partir de uma abordagem quantitativa, com base nas respostas obtidas por meio do questionário aplicado durante a coleta de dados. Os dados foram organizados em planilhas eletrônicas utilizando a ferramenta Planilhas Google (Google Sheets), possibilitando o cálculo das frequências absolutas e relativas, além da construção de gráficos para melhor visualização dos resultados.

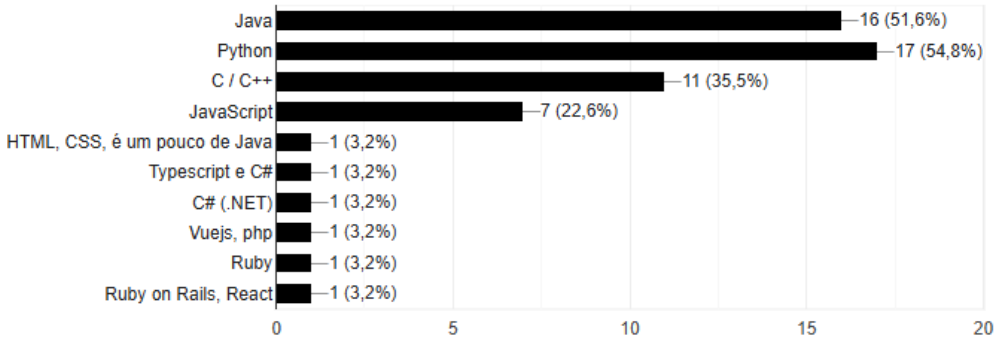
A interpretação dos dados foi conduzida com base nos eixos temáticos definidos na estrutura do questionário, permitindo uma análise segmentada e alinhada aos objetivos da pesquisa.

5.2.2.1 Perfil dos participantes

A análise do perfil dos participantes revelou que a maioria dos respondentes é composta por estudantes de Tecnologia ou Ciência da Computação, representando 50% (n=16) do total. Os profissionais atuantes na área correspondem a 43,8% (n=14), enquanto professores de tecnologia constituem 6,3% (n=2) dos participantes. Com base nesses dados, podemos inferir que a maior parte dos participantes possui experiência na área de tecnologia predominantemente superior a três anos, com destaque para participantes com mais de cinco anos de experiência.

Em relação às linguagens de programação, observou-se predominância de Python e Java, seguida por outras linguagens, como C/C++, JavaScript e outras. conforme demonstrado no gráfico abaixo.

Figura 2 – Linguagens de programação mais utilizadas.



Fonte: Autoria Própria (2026).

5.2.2.2 Dificuldades iniciais com programação

Ao serem questionados acerca de dificuldades enfrentadas no primeiro contato com programação, 56,2% (n=18) dos participantes afirmaram ter enfrentado dificuldades significativas, enquanto 34,4% (n=11) relataram dificuldades parciais. Apenas 9,4% (n=3) indicaram não ter enfrentado obstáculos relevantes nesse processo.

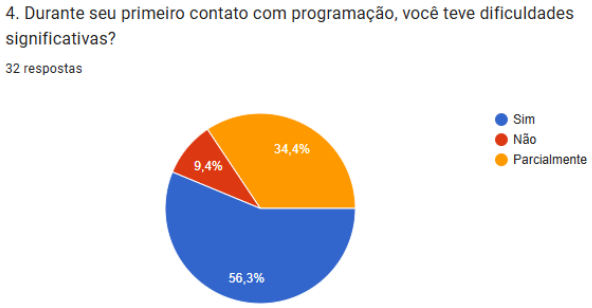
Esses dados demonstram que o aprendizado inicial de programação continua sendo um desafio significativo para a maioria dos participantes. Esse resultado está alinhado às evidências encontradas durante o levantamento bibliográfico, que apontam a programação introdutória como uma das disciplinas com maior índice de dificuldades, em razão da necessidade de desenvolver habilidades de abstração, decomposição de problemas e raciocínio lógico. Nesse contexto, estudos como os de Feitosa, Silva e Fabri (2021) e Paez, Fontdevila e Zangara (2024) sugerem que práticas baseadas em testes podem contribuir para estruturar melhor o processo de aprendizagem, incentivando o estudante a compreender os requisitos e o comportamento esperado das soluções antes da implementação do código.

5.2.2.3 Contato inicial com testes de software

Os dados indicam que 93,8% (n=30) dos participantes não tiveram oportunidade de contato com testes de software durante as disciplinas introdutórias de programação, enquanto apenas 6,2% (n=2) afirmaram ter tido essa experiência.

Esse resultado evidencia que o ensino inicial de programação ainda apresenta limitada integração com práticas voltadas à qualidade de software e testes automatizados. Tal evidência converge com os estudos analisados durante o levantamento bibliográfico, especialmente os de Feitosa, Silva e Fabri (2021) e Araujo, Flach e Bittencourt (2024), que defendem a introdução

Figura 3 – Dificuldade inicial no contato com programação.



Fonte: Autoria Própria (2026).

precoce de testes como estratégia para favorecer a compreensão dos problemas, proporcionar *feedback* contínuo e estimular a autonomia dos estudantes durante o processo de aprendizagem.

5.2.2.4 Uso de testes automatizados

Sobre o uso de testes automatizados durante o desenvolvimento de software, 37,5% (n=12) dos participantes afirmaram utilizá-los apenas "às vezes", enquanto 28,1% (n=9) relataram nunca utilizá-los. Outros 18,8% (n=6) responderam que os utilizam raramente, 12,5% (n=4) frequentemente e apenas 3,1% (n=1) declararam utilizá-los sempre.

Os resultados demonstram que a utilização de testes automatizados ainda é pouco consolidada entre os participantes da pesquisa. Considerando que 84,4% dos respondentes utilizam testes apenas ocasionalmente, raramente ou nunca, observa-se que essa prática ainda não faz parte da rotina de desenvolvimento da maioria dos participantes, mesmo entre aqueles que já atuam profissionalmente na área de Tecnologia da Informação.

Essa baixa aderência pode estar associada a diferentes fatores. O primeiro deles refere-se ao reduzido contato com testes de software durante a formação acadêmica, evidenciado nesta pesquisa pelo fato de 93,8% dos participantes afirmarem não ter tido acesso ao tema nas disciplinas introdutórias de programação. Além disso, conforme discutido por Paez, Fontdevila e Zangara (2024), a adoção de práticas baseadas em testes depende de uma mudança de mentalidade no processo de desenvolvimento, exigindo que os desenvolvedores incorporem atividades de validação desde as etapas iniciais da implementação. Essa mudança tende a representar uma barreira, sobretudo para profissionais e estudantes acostumados a modelos tradicionais de desenvolvimento.

Outro aspecto relevante diz respeito à percepção de custo inicial associada aos testes automatizados. Embora sua utilização contribua para aumentar a confiabilidade do software

e reduzir defeitos ao longo do ciclo de desenvolvimento, muitos desenvolvedores tendem a considerar a escrita de testes como uma atividade que demanda tempo adicional e cujo retorno não é imediatamente percebido. Esse comportamento é frequentemente observado em ambientes com prazos reduzidos ou em contextos onde a qualidade do software ainda não é tratada como requisito estratégico, favorecendo a implementação direta das funcionalidades em detrimento da construção de uma suíte de testes.

Por outro lado, os resultados obtidos nesta pesquisa indicam que, apesar da baixa utilização prática, os participantes reconhecem os benefícios proporcionados pelo TDD e pelos testes automatizados para a organização das soluções, compreensão dos requisitos e melhoria da qualidade do código. Essa aparente contradição evidencia a existência de uma lacuna entre o reconhecimento dos benefícios da metodologia e sua efetiva aplicação, sugerindo que iniciativas voltadas à inserção gradual de testes automatizados durante a formação acadêmica podem contribuir para ampliar sua adoção tanto no ambiente educacional quanto no contexto profissional.

5.2.2.5 Familiaridade com o TDD

Em relação ao grau de familiaridade com a metodologia TDD, 41,9%(n=13) dos participantes indicaram conhecê-la, embora a apliquem apenas de forma ocasional. Outros 38,7% (n=12) relataram já ter ouvido falar sobre o TDD, porém nunca o utilizaram na prática. Além disso, 16,1% (n=5) afirmaram não possuir conhecimento sobre a metodologia, enquanto apenas 3,1% (n=1) declararam aplicá-la de forma frequente.

Esses dados evidenciam uma diferença importante entre exposição, conhecimento conceitual e prática efetiva. O grupo que apenas ouviu falar sobre o TDD representa um contato inicial e superficial com a metodologia, sem necessariamente compreender seus princípios ou etapas de aplicação. Já os participantes que afirmaram conhecê-la, mas utilizá-la apenas ocasionalmente, indicam a existência de um conhecimento conceitual que ainda não se converteu em prática recorrente. Por fim, o número reduzido de respondentes que aplicam o TDD com frequência demonstra que a metodologia ainda possui baixa adoção efetiva entre os participantes da pesquisa.

Essa lacuna entre conhecer a metodologia e aplicá-la na prática é consistente com os achados de Paez, Fontdevila e Zangara (2024), que destacam que a adoção do TDD exige mais do que a compreensão teórica de seus conceitos. Segundo os autores, fatores como curva

de aprendizagem inicial, necessidade de mudança de mentalidade para uma abordagem *test-first*, domínio prévio de testes automatizados e oportunidades de prática contínua influenciam diretamente a incorporação efetiva da metodologia. Assim, os dados sugerem que a simples apresentação conceitual do TDD não é suficiente para promover sua adoção, sendo necessário criar experiências práticas e progressivas durante a formação acadêmica.

5.2.2.6 Aplicação prática do TDD

Ao serem questionados acerca da aplicação prática do TDD, 50% (n=16) dos entrevistados afirmaram nunca ter utilizado essa metodologia. Por outro lado, 34,4% (n=11) relataram ter aplicado o TDD em ambientes acadêmicos, enquanto 15,6% (n=5) mencionaram sua utilização no âmbito profissional.

Os resultados demonstram que o primeiro contato com o TDD ocorreu predominantemente em ambientes educacionais. Considerando que a amostra foi composta por estudantes, profissionais da área de Tecnologia da Informação e docentes, observa-se que, mesmo entre aqueles que já atuam profissionalmente, a experiência prática com a metodologia permanece reduzida. Esse resultado reforça os desafios apontados por Paez, Fontdevila e Zangara (2024), que destacam a baixa adoção do TDD em contextos de ensino e a necessidade de estratégias pedagógicas capazes de aproximar os estudantes das práticas consolidadas pela indústria de software.

5.2.2.7 Percepção sobre o impacto do TDD

Ao serem questionados acerca do impacto do TDD no processo de desenvolvimento de software, 46,9% (n=15) dos profissionais afirmaram que essa metodologia demanda um maior esforço inicial, contudo, traz benefícios ao longo do projeto. Outros 28,1% (n=9) destacaram que o TDD auxilia na melhor estruturação da solução do problema, enquanto 25% (n=8) apontaram a aprimoramento da qualidade do código como principal vantagem.

Os dados indicam que os participantes reconhecem benefícios relevantes na adoção do TDD, sobretudo relacionados à organização do desenvolvimento e à melhoria da qualidade do software. Essa percepção está em consonância com os estudos revisados, que apontam o TDD como uma prática capaz de promover maior compreensão dos requisitos, reduzir defeitos e incentivar o desenvolvimento incremental das soluções (Beck, 2003; Martin, 2019; Feitosa; Silva; Fabri, 2021).

5.2.2.8 Contribuição do TDD para compreensão e raciocínio lógico

No que diz respeito à contribuição do TDD para a compreensão do problema e dos requisitos, utilizando uma escala de 1 a 5, foi registrada uma média de 4,03. Além disso, verificou-se que 75% (n=24) dos participantes atribuíram notas iguais ou superiores a 4, indicando uma percepção favorável em relação ao impacto da metodologia nesse aspecto.

Esses resultados reforçam a hipótese investigada nesta pesquisa de que práticas baseadas em testes podem contribuir não apenas para a melhoria da qualidade do código, mas também para o desenvolvimento de competências cognitivas fundamentais no ensino de programação. Essa percepção está alinhada aos estudos de Feitosa, Silva e Fabri (2021), que associam a elaboração prévia de testes ao desenvolvimento do pensamento computacional, de Araujo, Flach e Bittencourt (2024), que evidenciam os benefícios do *feedback* imediato para a aprendizagem, e de Paez, Fontdevila e Zangara (2024), que destacam o potencial do TDD para estimular uma compreensão mais estruturada dos problemas antes da implementação das soluções.

De maneira geral, a análise dos resultados obtidos, em conjunto com as evidências identificadas durante o levantamento bibliográfico, permite responder à questão de pesquisa proposta neste trabalho: *Quais são as oportunidades e os desafios associados à aplicação do TDD no ensino introdutório de programação?* Os dados indicam que, embora existam desafios relacionados à baixa adoção da metodologia, à ausência de contato com testes de software durante a formação inicial e à necessidade de adaptação pedagógica, o TDD apresenta oportunidades relevantes para fortalecer o raciocínio lógico, melhorar a compreensão dos requisitos e aproximar o ensino das práticas consolidadas pela indústria de software. Dessa forma, os resultados obtidos fornecem subsídios para a proposta de implantação gradual apresentada neste trabalho, fundamentando-a tanto nas evidências da literatura quanto nas percepções dos participantes da pesquisa.

5.2.3 Discussão da Pesquisa

Diante do contexto apresentado e das evidências identificadas na literatura acerca dos desafios do ensino introdutório de programação e do potencial das práticas baseadas em testes, emerge a seguinte questão de pesquisa que orienta este estudo:

Quais são as oportunidades e os desafios associados à aplicação do TDD no ensino introdutório de programação?

A análise integrada do levantamento bibliográfico e dos dados obtidos por meio do *survey* indica que a adoção do TDD no ensino introdutório de programação ainda enfrenta desafios importantes. Entre eles destacam-se as dificuldades inerentes ao aprendizado inicial da programação, a baixa inserção de conteúdos relacionados a testes de software nas disciplinas introdutórias e a reduzida experiência prática dos estudantes com metodologias baseadas em testes.

Os resultados do *survey* evidenciaram que a maioria dos participantes relatou dificuldades durante o primeiro contato com programação, indicando que o desenvolvimento do raciocínio lógico, da abstração e da decomposição de problemas permanece como um dos principais obstáculos da formação em Computação. Esse cenário é consistente com os estudos analisados durante o levantamento bibliográfico, especialmente os de Feitosa, Silva e Fabri (2021), que destacam a necessidade de estratégias capazes de estruturar o processo de resolução de problemas desde os primeiros contatos dos estudantes com a programação.

Outro desafio identificado refere-se à reduzida inserção de testes de software no processo formativo. O fato de 93,8% (n=30) dos participantes afirmarem não ter tido contato com testes durante as disciplinas introdutórias evidencia uma lacuna entre a formação acadêmica e as práticas amplamente utilizadas na indústria de software. Essa constatação reforça os resultados apresentados por Araujo, Flach e Bittencourt (2024), Feitosa, Silva e Fabri (2021) e Paez, Fontdevila e Zangara (2024), que defendem a inserção precoce de testes automatizados como estratégia para favorecer a aprendizagem e desenvolver competências relacionadas à qualidade de software.

Os dados também revelaram uma diferença significativa entre conhecer o TDD e utilizá-lo efetivamente. Embora uma parcela expressiva dos participantes tenha declarado conhecer ou já ter ouvido falar sobre a metodologia, poucos afirmaram aplicá-la de forma recorrente. Esse resultado confirma os achados de Paez, Fontdevila e Zangara (2024), segundo os quais a adoção do TDD depende não apenas da compreensão de seus conceitos, mas também da existência de oportunidades práticas de aplicação, mudança de mentalidade para uma abordagem *test-first* e desenvolvimento contínuo de habilidades relacionadas aos testes automatizados.

Apesar dos desafios observados, os resultados da pesquisa evidenciam diversas oportunidades para a utilização do TDD como estratégia de apoio ao ensino de programação introdutória. Os participantes demonstraram uma percepção predominantemente positiva quanto aos benefícios da metodologia, reconhecendo sua contribuição para a compreensão dos requisitos,

organização das soluções e melhoria da qualidade do código.

Essa percepção encontra respaldo na literatura analisada. Conforme discutido por Feitosa, Silva e Fabri (2021), a elaboração prévia de testes estimula o planejamento da solução antes da implementação, reduzindo abordagens baseadas exclusivamente em tentativa e erro. De forma semelhante, Araujo, Flach e Bittencourt (2024) demonstram que mecanismos de testes automatizados e *feedback* imediato favorecem a aprendizagem ao permitir que os estudantes validem continuamente suas soluções e compreendam seus erros durante o processo de desenvolvimento.

Os resultados obtidos nas questões de percepção também reforçam esse potencial. As médias elevadas atribuídas pelos participantes às contribuições do TDD para compreensão dos problemas, entendimento dos requisitos e desenvolvimento do raciocínio lógico indicam que a metodologia é percebida não apenas como uma prática voltada à qualidade do software, mas também como um instrumento capaz de apoiar competências cognitivas essenciais ao processo de aprendizagem da programação.

A integração entre os resultados obtidos e o levantamento bibliográfico evidencia importantes implicações pedagógicas para o ensino de programação introdutória. Em vez de tratar os testes apenas como mecanismo de validação do software após sua implementação, o TDD permite que eles sejam incorporados ao próprio processo de construção do conhecimento, incentivando os estudantes a refletirem sobre os requisitos e o comportamento esperado das soluções antes da escrita do código.

Nesse sentido, a metodologia favorece uma aprendizagem mais ativa e incremental, aproximando o ambiente acadêmico das práticas consolidadas pela indústria de software. A utilização de ciclos contínuos de desenvolvimento, validação, refatoração e *feedback* imediato contribui para reduzir a distância entre teoria e prática, permitindo que os estudantes desenvolvam competências técnicas e cognitivas de maneira integrada.

Esses resultados sustentam a proposta de implantação gradual apresentada neste trabalho, cuja estrutura busca introduzir conceitos relacionados a testes de software e TDD de forma progressiva, respeitando o nível de maturidade dos estudantes e reduzindo as barreiras identificadas tanto na literatura quanto no *survey*.

De forma geral, a integração entre o levantamento bibliográfico e os resultados empíricos permite concluir que o TDD apresenta potencial para contribuir significativamente com o ensino introdutório de programação. Entretanto, sua efetiva adoção depende de mudanças na organização das disciplinas, da capacitação docente, da inserção gradual de conteúdos

relacionados a testes de software e da oferta de oportunidades práticas para que os estudantes possam desenvolver familiaridade com a metodologia.

Assim, os resultados obtidos respondem à questão de pesquisa proposta ao demonstrar que as principais oportunidades associadas ao uso do TDD estão relacionadas ao fortalecimento do pensamento computacional, à melhoria da qualidade do código e à aproximação entre formação acadêmica e mercado de trabalho. Por outro lado, os principais desafios concentram-se na ausência de contato prévio com testes de software, na baixa adoção prática da metodologia e na necessidade de estratégias pedagógicas que promovam sua integração ao ensino de programação introdutória.

5.3 Proposta da abordagem baseada em TDD

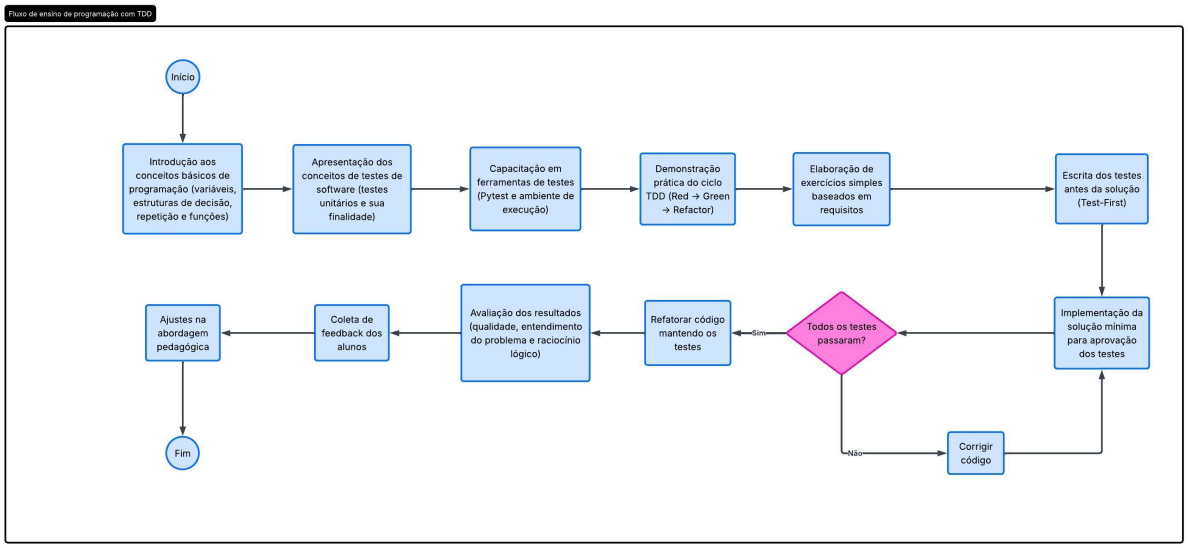
A adoção de práticas de qualidade de software desde os primeiros contatos com a programação pode contribuir para uma aprendizagem mais estruturada e alinhada às demandas do mercado de tecnologia. Nesse contexto, o TDD apresenta-se como uma abordagem promissora, pois incentiva a validação contínua do código, o raciocínio orientado a requisitos e o desenvolvimento incremental de soluções.

Com base nos desafios identificados na revisão da literatura e nos resultados do *survey* realizado com estudantes, docentes e profissionais da área de tecnologia, foi elaborada uma proposta de implantação gradual do TDD no ensino introdutório de programação. O objetivo da abordagem é introduzir os conceitos de testes e desenvolvimento orientado a testes de forma progressiva, respeitando o nível de maturidade dos estudantes e evitando sobrecarga cognitiva nos estágios iniciais da aprendizagem.

A proposta apresentada foi organizada em etapas sequenciais e cumulativas, nas quais cada fase prepara o estudante para o avanço ao próximo nível de maturidade técnica e conceitual. O fluxo busca introduzir práticas de qualidade de software, sem substituir os fundamentos tradicionais da programação, promovendo uma adoção gradual do desenvolvimento orientado a testes.

1. **Introdução aos conceitos básicos de programação** A primeira etapa concentra-se na consolidação dos conteúdos fundamentais da programação introdutória, incluindo variáveis, estruturas condicionais, estruturas de repetição e funções. O objetivo desse momento é garantir que os estudantes adquiram um domínio mínimo sobre construção de algoritmos e implementação de soluções antes da introdução de conceitos relacionados a testes. Essa

Figura 4 – Fluxo de Ensino de programação com TDD



Fonte: Autoria Própria (2026).

preparação inicial reduz barreiras cognitivas e evita que o uso do TDD seja percebido como um elemento adicional de complexidade.

2. **Apresentação dos conceitos de testes de software** Após o domínio dos conceitos básicos, ocorre a introdução aos fundamentos de testes de software, com foco em testes unitários e sua finalidade dentro do ciclo de desenvolvimento. Nesta etapa, busca-se demonstrar que testar não representa apenas validar erros, mas também especificar comportamentos esperados e aumentar a confiabilidade das soluções implementadas.
3. **Capacitação em ferramentas de testes** Nesta fase, os estudantes passam a conhecer ferramentas que permitem automatizar o processo de validação do código, utilizando recursos como o framework *Pytest* e ambientes apropriados para a execução dos testes. O objetivo é reduzir barreiras técnicas e fornecer uma infraestrutura simples para que os alunos concentrem seus esforços na aprendizagem dos conceitos, e não na configuração do ambiente.
4. **Demonstração prática do ciclo TDD** Com os conceitos introdutórios consolidados, é apresentado o ciclo clássico do TDD, composto pelas etapas Red, Green e Refactor. Inicialmente, cria-se um teste que falha (Red); posteriormente, desenvolve-se o código mínimo necessário para aprovação (Green); e, por fim, realiza-se a melhoria estrutural da solução, mantendo os testes válidos (Refactor). O objetivo desta etapa é demonstrar o funcionamento da abordagem em cenários controlados.
5. **Elaboração de exercícios simples baseados em requisitos** Após a demonstração do

ciclo, os estudantes recebem exercícios de baixa complexidade, formulados a partir de requisitos explícitos. O foco passa a ser interpretar necessidades antes da implementação da solução, estimulando a leitura crítica dos problemas e o desenvolvimento da capacidade de abstração.

6. **Escrita dos testes antes da solução (*Test-First*)** Nesta etapa, ocorre a introdução efetiva da prática central do TDD: elaborar os testes antes do desenvolvimento da funcionalidade. O estudante deve refletir sobre as entradas, saídas esperadas e critérios de aceitação antes de iniciar a implementação, fortalecendo o raciocínio orientado ao comportamento esperado do sistema.
7. **Implementação da solução mínima** Após definir os testes, o estudante implementa apenas o código necessário para alcançar sua aprovação. O objetivo é incentivar soluções incrementais e evitar implementações excessivamente complexas ou prematuras.
8. **Validação dos testes e correção** Concluída a implementação inicial, ocorre a execução dos testes automatizados. Caso os testes falhem, o estudante retorna ao código para realizar correções e ajustes até atingir o comportamento esperado. Esse ciclo iterativo favorece o aprendizado baseado em tentativa orientada e validação contínua.
9. **Refatoração mantendo os testes** Quando os testes são aprovados, inicia-se a etapa de refatoração. Nesse momento, busca-se melhorar a legibilidade, organização e a manutenção do código, sem alterar seu comportamento funcional. Os testes atuam como mecanismo de segurança para garantir a estabilidade durante as alterações.
10. **Avaliação dos resultados** Após a conclusão do exercício, são avaliados aspectos relacionados à qualidade do código produzido, ao entendimento dos requisitos e ao desenvolvimento do raciocínio lógico. Essa análise busca verificar não apenas o funcionamento da solução, mas também o processo utilizado para sua construção.
11. **Coleta de *feedback* dos estudantes** Os estudantes são convidados a registrar percepções sobre dificuldades, benefícios e sugestões relacionadas ao uso da abordagem. Esse retorno permite compreender fatores que influenciam o processo de aprendizagem e identificar oportunidades de melhoria.
12. **Ajustes na abordagem pedagógica** Com base nos resultados observados e nos *feedbacks* coletados, realiza-se o refinamento da estratégia de ensino. Essa etapa garante que o modelo permaneça adaptável às características dos estudantes e possa evoluir continuamente.

A proposta de implantação gradual do TDD apresentada neste trabalho não tem como

finalidade substituir os métodos tradicionais de ensino de programação, mas complementar o processo formativo por meio da introdução progressiva de práticas relacionadas à qualidade de software. Ao distribuir os conceitos em etapas sucessivas e alinhadas ao nível de conhecimento dos estudantes, busca-se reduzir as barreiras iniciais de adoção e favorecer uma aprendizagem mais estruturada, orientada à resolução de problemas e à validação contínua das soluções desenvolvidas. Dessa forma, o modelo proposto representa uma alternativa para aproximar o ambiente acadêmico das práticas contemporâneas da indústria de software, mantendo o foco no desenvolvimento técnico e cognitivo dos estudantes.

5.3.1 Exemplo de aplicação da proposta em uma disciplina

Com o objetivo de demonstrar a viabilidade da abordagem proposta, apresenta-se um exemplo de aplicação em uma disciplina introdutória de programação, considerando um curso de graduação em Computação ou áreas correlatas.

A proposta foi planejada para uma disciplina com carga horária aproximada de 60 horas, distribuídas ao longo de um semestre letivo de quinze semanas, contemplando aulas expositivas, atividades práticas em laboratório e exercícios individuais.

A Tabela 2 apresenta uma sugestão de sequência didática para implantação gradual da abordagem.

Tabela 2 – Exemplo de aplicação da abordagem proposta

Semanas	**Conteúdo**	**Atividade proposta**
1–4	Fundamentos da programação	Variáveis, estruturas condicionais, repetição, funções e resolução de exercícios tradicionais.
5–6	Testes de software	Apresentação de testes unitários, casos de teste, critérios de entrada e saída, cobertura básica e ferramentas como *pytest*.
7–8	Introdução ao TDD	Demonstração do ciclo Red-Green-Refactor utilizando exemplos simples.
9–12	Aplicação prática	Resolução de exercícios utilizando abordagem *test-first*, desenvolvimento incremental e refatoração.
13–14	Projeto Integrador	Desenvolvimento de um pequeno sistema utilizando TDD durante todo o processo de implementação.
15	Avaliação e feedback	Discussão dos resultados, coleta de feedback dos estudantes e reflexão sobre a metodologia.

Fonte: Autoria Própria (2026).

5.3.1.1 Exemplo prático de aplicação do ciclo TDD

Para exemplificar a aplicação da abordagem proposta, considere um exercício introdutório frequentemente utilizado nas primeiras disciplinas de programação.

Problema proposto: Desenvolver uma função capaz de receber dois números inteiros e retornar sua soma.

Antes de implementar a solução, o estudante deve escrever um teste automatizado que represente o comportamento esperado da função. Utilizando o framework *pytest*, o teste pode ser definido da seguinte forma:

Código-fonte 1 – Teste inicial utilizando pytest

```
1 from calculadora import somar
2
3 def test_somar():
4     assert somar(2, 3) == 5
```

Nesse momento, a execução do teste resulta em falha, pois a função ainda não foi implementada. Essa etapa corresponde à fase *Red* do ciclo TDD, na qual o teste evidencia a ausência da funcionalidade esperada.

Na sequência, o estudante implementa apenas o código mínimo necessário para que o teste seja aprovado.

Código-fonte 2 – Implementação mínima da solução

```
1 def somar(a, b):
2     return a + b
```

Após executar novamente os testes, observa-se que o comportamento esperado foi atingido, caracterizando a etapa *Green*. Em seguida, inicia-se a fase de *Refactor*, na qual o estudante pode melhorar a organização do código sem alterar seu comportamento externo.

Embora a função apresentada seja simples e não exija alterações significativas, o professor pode utilizar esse momento para discutir boas práticas de programação, como documentação da função, padronização de nomenclatura, organização do projeto e inclusão de novos casos de teste.

Por exemplo, novos testes podem ser adicionados para verificar diferentes cenários:

Código-fonte 3 – Ampliação dos casos de teste

```
1 def test_somar_negativos():
```

```
    assert somar(-2, -3) == -5
def test_somar_zero():
    assert somar(5, 0) == 5
```

Com isso, o estudante percebe que a criação dos testes orienta o desenvolvimento da solução, permitindo validar continuamente o comportamento do programa e evoluir o código de maneira incremental. À medida que a disciplina avança, exercícios envolvendo estruturas condicionais, repetições, listas, orientação a objetos e manipulação de arquivos podem seguir a mesma dinâmica, mantendo o princípio de elaborar os testes antes da implementação da funcionalidade.

Durante as atividades práticas, recomenda-se a utilização da linguagem Python em conjunto com o framework *pytest*, em função de sua simplicidade e ampla utilização em disciplinas introdutórias. Como ambiente de desenvolvimento podem ser utilizados Visual Studio Code, PyCharm Community ou Google Colab, desde que permitam a execução dos testes automatizados. Além disso, podem ser empregados repositórios GitHub para disponibilização dos exercícios e ferramentas de integração contínua, como GitHub Actions, para automatizar a execução dos testes, quando pertinente.

Esse exemplo evidencia a aplicação prática das quatro fases da abordagem proposta. Na Fase 1, o estudante consolida os conceitos fundamentais de programação e conhece os princípios dos testes de software. Na Fase 2, o docente demonstra o ciclo *Red-Green-Refactor* utilizando um problema simples. Na Fase 3, os estudantes passam a elaborar seus próprios testes antes da implementação das soluções (*test-first*), validando e refinando o código de forma iterativa. Por fim, na Fase 4, o código é refatorado, os resultados são avaliados e os estudantes fornecem *feedback* sobre a atividade, permitindo aperfeiçoar continuamente a abordagem pedagógica.

Os exercícios devem evoluir gradativamente em complexidade. Inicialmente, os estudantes podem desenvolver funções simples, como operações matemáticas, verificação de números primos, cálculo de médias ou manipulação de listas. Em seguida, podem ser propostos problemas envolvendo arquivos, tratamento de exceções, orientação a objetos e pequenos projetos, sempre iniciando pela definição dos testes antes da implementação da solução.

A avaliação da aprendizagem pode considerar diferentes dimensões, contemplando não apenas o funcionamento correto do programa, mas também aspectos relacionados ao processo de desenvolvimento. Sugere-se que sejam avaliados critérios como:

- Corretude da solução proposta;
- Elaboração dos testes antes da implementação;
- Cobertura dos principais casos de teste;
- Qualidade e organização do código;
- Utilização adequada do ciclo *Red-Green-Refactor*;
- Capacidade de interpretar requisitos e justificar as soluções adotadas.

Como materiais de apoio, recomenda-se a utilização de apostilas sobre testes unitários, exemplos comentados de TDD, vídeos demonstrativos do ciclo *Red-Green-Refactor*, roteiros de laboratório e exercícios progressivos. Esses recursos auxiliam na redução da curva de aprendizagem apontada pela literatura e favorecem a incorporação gradual da metodologia pelos estudantes.

6 CONCLUSÕES E TRABALHOS FUTUROS

O ensino de programação introdutória apresenta desafios significativos relacionados à compreensão de conceitos abstratos, ao desenvolvimento do raciocínio lógico e à resolução estruturada de problemas. Diante desse cenário, este trabalho teve como objetivo analisar a aplicabilidade e os potenciais benefícios da integração da metodologia TDD ao ensino introdutório de programação, considerando tanto as evidências identificadas na literatura quanto a percepção de estudantes, docentes e profissionais da área de tecnologia.

Por meio da revisão da literatura, foi possível identificar que a utilização de testes automatizados e do TDD pode contribuir para a melhoria da qualidade do código, para o fortalecimento do pensamento computacional e para o desenvolvimento de habilidades importantes no processo de aprendizagem. Os estudos analisados também evidenciaram desafios relacionados à adoção da metodologia, como a necessidade de mudança de mentalidade, a curva de aprendizagem inicial e a falta de contato prévio com práticas de testes.

Os resultados obtidos por meio do *survey* reforçaram essas evidências. Verificou-se que a maioria dos participantes enfrentou dificuldades durante o aprendizado inicial de programação e que o contato com testes de software nas disciplinas introdutórias ainda é bastante reduzido. Por outro lado, os respondentes demonstraram uma percepção positiva em relação ao TDD, acreditando em uma potencial contribuição para a compreensão dos requisitos, organização das soluções, melhoria da qualidade do código e desenvolvimento do raciocínio lógico.

Com base nos resultados obtidos, foi elaborada uma proposta de implantação gradual do TDD no ensino introdutório de programação, buscando introduzir conceitos de testes e desenvolvimento orientado a testes de forma progressiva e compatível com o nível de conhecimento dos estudantes. A proposta foi construída a partir das evidências levantadas na literatura e das percepções identificadas junto aos participantes da pesquisa.

Dessa forma, conclui-se que o TDD apresenta potencial para atuar como uma estratégia complementar no ensino de programação introdutória, contribuindo para uma aprendizagem mais estruturada, reflexiva e alinhada às práticas contemporâneas da indústria de *software*. Embora existam desafios para sua adoção, os resultados obtidos indicam que a integração gradual dessa abordagem pode representar uma oportunidade para aproximar a formação acadêmica das demandas do mercado e promover o desenvolvimento de competências técnicas e cognitivas relevantes para os futuros profissionais da área de tecnologia.

6.1 Limitações da Pesquisa

Apesar de os objetivos propostos terem sido alcançados, este estudo apresenta algumas limitações que devem ser consideradas na interpretação dos resultados.

A primeira refere-se ao tamanho e à composição da amostra utilizada na pesquisa. O *survey* contou com a participação de 32 respondentes, distribuídos entre estudantes, docentes e profissionais da área de Tecnologia da Informação. Embora essa amostra tenha permitido identificar tendências e percepções relevantes, seus resultados não podem ser generalizados para toda a comunidade acadêmica ou para o mercado de desenvolvimento de software.

Outra limitação está relacionada à natureza da pesquisa. Os resultados foram obtidos a partir de um levantamento bibliográfico e de um *survey*, refletindo as percepções e experiências dos participantes. Dessa forma, não foi possível avaliar experimentalmente os impactos da utilização do TDD sobre indicadores objetivos, como desempenho acadêmico, qualidade do código produzido ou evolução do raciocínio lógico dos estudantes.

Além disso, a abordagem gradual proposta neste trabalho possui caráter conceitual e foi elaborada com base na integração entre as evidências identificadas na literatura e os resultados obtidos na pesquisa. Sua aplicação em sala de aula não foi realizada no âmbito deste estudo, sendo necessária sua validação em contextos reais de ensino para verificar sua efetividade e identificar possíveis adaptações.

Por fim, o levantamento bibliográfico concentrou-se em estudos relacionados ao ensino de programação introdutória e ao uso do TDD nesse contexto, não contemplando outras metodologias de ensino ou práticas de qualidade de software que também possam contribuir para o processo de aprendizagem.

Apesar dessas limitações, acredita-se que os resultados apresentados oferecem evidências relevantes sobre as oportunidades e os desafios da utilização do TDD no ensino introdutório de programação, contribuindo para ampliar as discussões sobre a integração de práticas de qualidade de software à formação inicial em Computação.

6.2 Trabalhos Futuros

Como continuidade desta pesquisa, sugere-se a aplicação prática da abordagem proposta em disciplinas introdutórias de programação, permitindo avaliar seus impactos reais no processo de aprendizagem dos estudantes. A implementação da metodologia em ambientes educacionais

possibilitará analisar aspectos como desempenho acadêmico, desenvolvimento do raciocínio lógico, compreensão de requisitos, qualidade do código produzido e percepção dos alunos sobre a utilização de testes durante o aprendizado.

Além disso, futuras pesquisas poderão realizar estudos comparativos entre turmas que utilizem a abordagem proposta e turmas submetidas a metodologias tradicionais de ensino, buscando identificar diferenças significativas nos resultados obtidos. Também se recomenda a investigação da aplicação da metodologia em diferentes linguagens de programação, níveis de ensino e contextos institucionais, ampliando a compreensão sobre sua adaptabilidade e efetividade.

Outra possibilidade consiste em desenvolver materiais didáticos, roteiros de atividades, exercícios práticos e ferramentas de apoio voltados especificamente para a introdução do TDD em disciplinas de programação. Essas iniciativas poderão auxiliar docentes na implementação da abordagem e reduzir as barreiras identificadas ao longo desta pesquisa.

Por fim, recomenda-se a validação da proposta apresentada neste trabalho por meio de estudos experimentais ou quase-experimentais conduzidos em disciplinas introdutórias de programação. Nesses estudos, sugere-se a utilização de um grupo experimental, que adote a abordagem baseada em TDD, e de um grupo controle, submetido ao método tradicional de ensino. A aplicação de pré-testes e pós-testes permitirá comparar a evolução dos estudantes em aspectos como desempenho acadêmico, desenvolvimento do raciocínio lógico, compreensão de requisitos, qualidade do código produzido e percepção sobre o processo de aprendizagem. Além disso, poderão ser utilizados indicadores complementares, como taxa de aprovação, número de defeitos identificados, cobertura de testes, tempo de resolução dos exercícios e satisfação dos estudantes com a metodologia. A realização de investigações dessa natureza poderá fornecer evidências empíricas mais robustas sobre a efetividade da proposta, contribuindo para sua consolidação como estratégia pedagógica para o ensino introdutório de programação.

REFERÊNCIAS

ADMINISTRATION, F. A. **Summary of the FAA's Review of the Boeing 737 MAX**. 2020. Disponível em: https://www.faa.gov/sites/faa.gov/files/2022-08/737$_{R}TS_{S}ummary.pdf.Acessoem19out$.2025.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 2. ed. Cambridge: Cambridge University Press, 2016.

ARAUJO, L.; FLACH, C.; BITTENCOURT, R. Qual tipo de feedback ajuda os estudantes novatos em programação? um estudo exploratório com múltiplos feedbacks em um curso introdutório de python. In: Simpósio Brasileiro de Educação em Computação, 4., 2021, Porto Alegre, **Anais** [...]. Porto Alegre: SBC, 2024. p. 42–51.

AZEVEDO, D. d. S. N.; SILVA, F. d. S. **Pensamento Computacional no Ensino Fundamental** : uma análise da abordagem na bncc e suas implicações na área da matemática. 2025. Acesso em 12 nov. 2025. Disponível em: https://periodicos.utfpr.edu.br/rtr/article/view/20027/10748.

BAUMGARTNER, C. **Cuidado! Saiba por que a ausência de testes de software adequados pode arruinar seu projeto!** 2025. Disponível em: https://testingcompany.com.br/blog/cuidado-saiba-por-que-a-ausencia-de-testes-de-software-adequados-pode-arruinar-seu-projeto. Acesso em 19 out. 2025.

BECK, K. **Test-Driven Development: By Example**. Boston: Addison-Wesley, 2003.

BRASIL. **Educação digital e midiatrica**: como elaborar e implementar i cyrrpiculo nas escolas. 2025. Disponível em: https://www.gov.br/mec/pt-br/escolas-conectadas/BNCCComputaoCompletodiagramado.pdfpage=66.07. Acesso em 12 nov. 2025.

COMPUTER SCIENCE TEACHERS ASSOCIATION. **CSTA K-12 Computer Science Standards**. 2011. Disponível em: https://ioi.te.lv/conf/c5$_{C}STA.pdf.Acessoem$24set.2025.

CONBOY, K. Agility from first principles: Reconstructing the concept of agility in information systems development. **Information Systems Research**, v. 20, n. 3, p. 329–354, 2009.

DARGAINS, A. R.; SAMPAIO, F. F. Ensino de programação, robótica educacional e taxonomia de bloom. 2020. Acesso em: 21 jul. 2025. Disponível em: https://econtents.sbu.unicamp.br/inpec/index.php/tsc/article/view/14702.

DOUCE, C.; LIVINGSTONE, D.; ORWELL, J. Automatic test-based assessment of programming: A review. **Journal on Educational Resources in Computing**, v. 5, n. 3, 2005.

FEITOSA, Y. G.; SILVA, M. A. G.; FABRI, J. Desenvolvimento baseado em testes com aplicação de critérios de teste no ensino introdutório de computação. In: Simpósio Brasileiro de Educação em Computação, 1., 2021, Porto Alegre, **Anais** [...]. Porto Alegre: SBC, 2021.

IHANTOLA, P. *et al.* Review of recent systems for automatic assessment of programming assignments. **Computer Science Education**, v. 20, n. 2, p. 113–142, 2010.

KOBAK, A. **The 2021 Facebook Outage**. 2024. Disponível em: https://www.linkedin.com/pulse/2021-facebook-outage-aykut-kobak-cjmgf/. Acesso em 19 out. 2025.

LOPES, R. P.; FüRKOTTER, M. Formação inicial de professores em tempos de tdic: uma questão em aberto. 2016. Acesso em 21 jul. 2025. Disponível em: https://doi.org/10.1590/0102-4698150675.

MARTIN, R. C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Boston: Prentice Hall, 2019.

MARTINS, J. D. Experimentos em síntese de alto nível orientada à minimização de área e potência. 2019. Acesso em 9 set. 2025. Disponível em: https://lume.ufrgs.br/handle/10183/200169.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3. ed. Indianapolis: John Wiley & Sons, 2011.

PAEZ, N.; FONTDEVILA, D.; ZANGARA, A. Test-driven development: Challenges and recommendations for trainers and educators. In: . [s.n.], 2024. Acesso em 21 jul. 2025. Disponível em: https://ieeexplore.ieee.org/document/10700153.

PRESSMAN, R. S.; MAXIM, B. R. **Software Engineering: A Practitioner's Approach**. 8. ed. New York: McGraw-Hill Education, 2016.

ROCHA, R. G. d. C. *et al.* **Uma abordagem prática de design thinking no processo de ensino-aprendizagem na engenharia de software** . 2021. Acesso em 21 jul. 2025. Disponível em: https://doi.org/10.22456/1679-1916.121368.

SOMMERVILLE, I. **Software Engineering**. 9. ed. Boston: Addison-Wesley, 2011.

APÊNDICE A – QUESTIONÁRIO

Formulário de Pesquisa: Experiência com o Ensino de Programação e Abordagens Baseadas em Testes

Termo de Consentimento Livre e Esclarecido (TCLE) para Participação em Pesquisa

Título da Pesquisa: *"Uma Abordagem Baseada em Testes para o Ensino de Programação Introdutória"*
Pesquisador Responsável: Jefersson Francisco Gameleira Pereira
Orientador: Ferdinandy Silva Chagas
Instituição: Universidade Federal Rural do Semi-Árido
E-mail do Pesquisador: jefersson.pereira@alunos.ufersa.edu.br
E-mail do Orientador: ferdinandy@ufersa.edu.br

Prezado(a) Participante,

Você está sendo convidado(a) a participar, como voluntário(a), da pesquisa intitulada *"Percepção de Alunos e Profissionais de Tecnologia sobre o Uso de Testes Automatizados e TDD no Ensino e na Prática de Programação"*.

Esta pesquisa tem como objetivo investigar a percepção de estudantes e profissionais da área de tecnologia sobre:

- As dificuldades no aprendizado inicial de programação;
- O uso de testes automatizados;
- A aplicação do Desenvolvimento Orientado a Testes (TDD);

Sua participação é totalmente voluntária. Antes de decidir participar, é importante que você compreenda as informações abaixo.

Benefícios Esperados

Sua participação contribuirá para a validação e aprimoramento de uma metodologia de ensino que pode beneficiar estudantes e profissionais interessados em aprender programação de forma estruturada e eficiente, com foco na qualidade de código.

Riscos e Desconfortos

Não são esperados riscos significativos associados à sua participação, uma vez que o estudo envolve apenas a resposta a um formulário de pesquisa.

Confidencialidade

Todas as informações coletadas serão mantidas estritamente confidenciais. Os dados serão armazenados de forma segura e utilizados exclusivamente para fins acadêmicos. A identidade dos participantes será preservada em todas as etapas da pesquisa, e os dados não serão compartilhados com terceiros.

Livre Retirada e Esclarecimentos

Sua participação é voluntária, e você pode desistir de participar do estudo a qualquer momento, sem qualquer penalização ou prejuízo. Se houver dúvidas ou se você precisar de mais esclarecimentos, entre em contato com os responsáveis pela pesquisa por meio dos e-mails fornecidos.

Declaração de Consentimento:
Ao assinar este termo, você indica que leu e compreendeu as informações apresentadas, está ciente dos procedimentos e concorda em participar voluntariamente desta pesquisa.

* Indica uma pergunta obrigatória

1. Li e concordo com o TCLE: *

Marcar apenas uma oval.

- ◯ Sim *Pular para a pergunta 2*
- ◯ Não

Informações Gerais:

2. 1. Você é:

Marcar apenas uma oval.

- ◯ Estudante de Tecnologia/Computação
- ◯ Profissional da área de tecnologia
- ◯ Professor(a)
- ◯ Outro: _____

3. 2. Tempo de experiência com programação:

Marcar apenas uma oval.

- ◯ Menos de 1 ano
- ◯ 1 a 3 anos
- ◯ 3 a 5 anos
- ◯ Mais de 5 anos

4. 3. Linguagens que você utiliza com maior frequência:

Marque todas que se aplicam.

- ☐ Java
- ☐ Python
- ☐ C / C++
- ☐ JavaScript
- ☐ Outro: _____

Experiência durante a Programação Introdutória

5. 4. Durante seu primeiro contato com programação, você teve dificuldades significativas?

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não
- ◯ Parcialmente

6. 5. Quais foram as principais dificuldades enfrentadas?

Marque todas que se aplicam.

- [ ] Lógica de programação
- [ ] Compreensão de erros
- [ ] Estruturação do código
- [ ] Falta de feedback claro
- [ ] Outro: _____

7. 6. Você aprendeu sobre testes de software já nas disciplinas iniciais?

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

Desenvolvimento Orientado a Testes (TDD)

Nesta seção buscamos compreender sua percepção sobre o **Desenvolvimento Orientado a Testes (Test-Driven Development – TDD)**, uma metodologia em que os testes são escritos antes da implementação do código, guiando o desenvolvimento da solução.

8. 7. Você costuma utilizar testes automatizados durante o desenvolvimento de software?

Marcar apenas uma oval.

- ◯ Sempre
- ◯ Frequentemente
- ◯ Às vezes
- ◯ Raramente
- ◯ Nunca

9. 8. Considerando a abordagem TDD, em que os testes são escritos antes da implementação do código, o quanto você acredita que essa prática ajuda na compreensão do problema e dos requisitos?

Marcar apenas uma oval.

	1	2	3	4	5	
Não	◯	◯	◯	◯	◯	Ajuda muito

10. 9. Na sua percepção, quais benefícios o uso do TDD pode trazer durante o desenvolvimento de software?

Marque todas que se aplicam.

- [ ] Melhor organização do código
- [ ] Maior confiança na funcionalidade implementada
- [ ] Melhor compreensão dos requisitos do problema
- [ ] Redução de erros e retrabalho
- [ ] Facilita a manutenção do código
- [ ] Não percebo benefícios

11. 10. Qual é o seu nível de familiaridade com o Test-Driven Development (TDD)?

Marcar apenas uma oval.

- ◯ Conheço e aplico com frequência
- ◯ Conheço, mas aplico raramente
- ◯ Já ouvi falar, mas nunca utilizei
- ◯ Não conheço a metodologia

12. 11. Você já aplicou TDD na prática?

Marcar apenas uma oval.

- ◯ Sim, em ambiente acadêmico
- ◯ Sim, em ambiente profissional
- ◯ Não

13. 12. Na sua percepção, qual das afirmações melhor representa o impacto do **TDD** no desenvolvimento de software?

Marcar apenas uma oval.

- ◯ Melhora a qualidade do código
- ◯ Ajuda a estruturar melhor a solução do problema
- ◯ Exige maior esforço inicial, mas traz benefícios posteriormente
- ◯ Torna o desenvolvimento mais lento
- ◯ Não percebo diferença significativa

14. 13. Em sua opinião, o uso da metodologia TDD contribui para o desenvolvimento do raciocínio lógico e da capacidade de resolução de problemas?

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Totalmente

Comentários Finais

15. 14. Caso queira, compartilhe algo mais sobre sua experiência com o aprendizado de programação:

Agradecimento

Agradecemos sua participação! Suas respostas contribuirão para o desenvolvimento de abordagens mais eficazes no ensino de programação.

Este conteúdo não foi criado nem aprovado pelo Google.

Google Formulários

A Tabela 3 apresenta a relação entre as questões do questionário aplicado e os objetivos específicos desta pesquisa. Essa rastreabilidade evidencia como cada questão contribuiu para responder à questão de pesquisa e alcançar os objetivos propostos.

Tabela 3 – Relação entre as questões do questionário e os objetivos específicos da pesquisa

Questão	**Tipo**	**Objetivo da questão**	**Objetivo específico relacionado**
1	Consentimento	Registrar o aceite do TCLE para participação na pesquisa.	Garantir os aspectos éticos da pesquisa.
2	Perfil	Identificar o perfil do participante (estudante, profissional ou docente).	Caracterizar a amostra da pesquisa.
3	Perfil	Identificar o tempo de experiência com programação.	Caracterizar o nível de experiência dos participantes.
4	Perfil	Identificar as linguagens de programação utilizadas.	Contextualizar o ambiente tecnológico dos respondentes.
5	Múltipla escolha	Verificar se houve dificuldades durante o aprendizado inicial de programação.	Identificar desafios presentes no ensino introdutório de programação.
6	Múltipla escolha	Identificar quais dificuldades foram enfrentadas durante o aprendizado.	Compreender fatores que dificultam o processo de aprendizagem.
7	Múltipla escolha	Verificar se os participantes tiveram contato com testes de software durante as disciplinas introdutórias.	Investigar a inserção de testes de software no ensino de programação.
8	Escala de frequência	Avaliar a frequência de utilização de testes automatizados.	Investigar a utilização prática de testes automatizados na formação e no mercado.
9	Escala Likert	Avaliar a percepção sobre a contribuição do TDD para compreensão do problema e dos requisitos.	Analisar a percepção sobre os benefícios pedagógicos do TDD.
10	Múltipla escolha	Identificar quais benefícios os participantes associam ao uso do TDD.	Identificar oportunidades de utilização do TDD no desenvolvimento de software.
11	Múltipla escolha	Avaliar o nível de familiaridade com a metodologia TDD.	Investigar o conhecimento conceitual sobre TDD.
12	Múltipla escolha	Verificar se houve aplicação prática do TDD e em qual contexto.	Analisar a adoção do TDD em ambientes acadêmicos e profissionais.
13	Múltipla escolha	Identificar a percepção sobre o impacto do TDD durante o desenvolvimento de software.	Investigar os benefícios e desafios percebidos na utilização do TDD.
14	Escala Likert	Avaliar a percepção sobre a contribuição do TDD para o desenvolvimento do raciocínio lógico e resolução de problemas.	Analisar o potencial pedagógico do TDD para o ensino introdutório de programação.
15	Resposta aberta	Permitir que os participantes relatem experiências, sugestões ou dificuldades não contempladas nas questões objetivas.	Complementar qualitativamente os resultados obtidos na pesquisa.

Fonte: Autoria Própria (2026).